DOCUMENT RESUME

ED 080 122 LI 004 427

AUTHOR	Kilgour, Frederick G., Comp.; Davis, Hillis D., Comp.
TITLE	The Development of a Computerized Regional Library System. Appendices 26 & 27. Final Report.
INSTITUTION	Ohio Coll. Library Center, Columbus.
SPONS AGENCY	Office of Education (DHEW), Washington, D.C.
BUREAU NO	BR-9-0554
PUB DATE	Jun 73
CONTRACT	OEC-0-70-2289(506)
NOTE	168p.; (0 References)
EDRS PRICE	MF-$0.65 HC-$6.58
DESCRIPTORS	Cataloging; College Libraries; *Computer Programs; Expenditure Per Student; Library Automation; *Library Expenditures; *Library Networks; Library Technical Processes; *On Line Systems; Regional Programs; Union Catalogs; *University Libraries
IDENTIFIERS	OCLC; *Ohio College Library Center

ABSTRACT

The purpose of the Ohio College Library Center (OCLC) computerized regional library system is to provide an on-line system that makes available to faculty and students in individual colleges and universities the library resources throughout a region, while at the same time decelerating the rate of rise of per-student library costs. The research and development culminated in the successful implementation of an on-line union catalog and shared cataloging system. The final report of the project is LI 004 422. This document contains appendices twenty-six and -seven: The Ohio College Library Center Program/Subroutine Documentation--(26) Generate Pack Definition Tables (GENPDTS) and (27) Catalog Card Format Program (CCFP). The FORTRAN program, GENPDTS, records the institutions' coded cataloging requirements in the format required by the CCFP. The second step in the formatting and production of catalog cards which are sent to member libraries is the CCFP. Depending on pre-determined specifications for each member library, CCFP creates a card for each catalog set up by the member, varying the headings and setting indentions as prescribed by the member. (Other appendices are LI 004 423 through LI 004 426 and LI 004 428.) (Author/SJ)

FILMED FROM BEST AVAILABLE COPY

ED 080122

U.S. DEPARTMENT OF HEALTH, EDUCATION & WELFARE
NATIONAL INSTITUTE OF EDUCATION
THIS DOCUMENT HAS BEEN REPRODUCED EXACTLY AS RECEIVED FROM THE PERSON OR ORGANIZATION ORIGINATING IT. POINTS OF VIEW OR OPINIONS STATED DO NOT NECESSARILY REPRESENT OFFICIAL NATIONAL INSTITUTE OF EDUCATION POSITION OR POLICY

Final Report

Project No. 9-0554
Contract No. OEC-0-70-2289 (506)

June 1973

THE DEVELOPMENT OF A COMPUTERIZED REGIONAL LIBRARY SYSTEM

APPENDICES 26 & 27

Frederick G. Kilgour
The Ohio College Library Center
1550 West Henderson Road
Columbus, Ohio 43220

Hillis D. Davis
Cooperative College Library Center
159 Forrest Avenue N.E.
Atlanta, Georgia 30303

LI 004 427

A P P E N D I C E S

I. Instruction Manual for Catalog Production. (LI 004 423)

II. Manual for OCLC Catalog Card Production; Revised and Enlarged. Judith Hopkins. (LI 004 423)

III. Creation of Machine Readable Catalog Entries; An Adaptation of the "Data Preparation Manual: MARC Editors." (LI 004 423)

IV. Cataloging on a Cathode Ray Tube Terminal. (LI 004 423)

V. Brief Description of the Serials Control System: A Preliminary Report. (LI 004 424)

VI. A Preliminary Description of the OCLC Serials Control System. (LI 004 424)

VII. Manual for Checking-In, Binding, and Claiming of Serials on a CRT Terminal - Draft of Preliminary Procedures. (LI 004 424)

VIII. Suggested Minimum Requirements for Serials Cataloging. (LI 004 424)

IX. OCLC Technical Processing System - A Preliminary Outline. (LI 004 424)

X. The Technical Processing System, May 1972. (LI 004 424)

XI. Recommended Standards for the Cataloging of Serials. (LI 004 424)

XII. Standards for Input Cataloging. (LI 004 424)

XIII. The Technical Processing System, August 1972. (LI 004 424)

XIV. Ohio College Library Center Annual Report, 1971/1972. (LI 004 424)

XV. Large On-Line Files of Bibliographic Data: An Efficient Design and a Mathematical Predictor of Retrieval Behavior. P.L. Long, K.B.L. Rastogi, J.E. Rush and J.A. Wyckoff. (Not Available EDRS)

XVI. OCLC Systems: Technical Aspects. Phillip Long. and Ohio State University Libraries Systems. Gerry D. Guthrie. (Not Available EDRS)

XVII. Name-Title Entry Retrieval from a MARC File. Philip L. Long and Frederick G. Kilgour. (Not Available EDRS)

XVIII. A Truncated Search Key Title Index. Philip L. Long and Frederick G. Kilgour. (Not Available EDRS)

XIX. Title-Only Entries Retrieved by Use of Truncated Search Keys. Frederick G. Kilgour, Philip L. Long, Eugene B. Leiderman and Alan L. Landgraf. (Not Available EDRS)

XX. Ohio College Library Center Systems. Frederick Kilgour. (Not Available EDRS)

XXI. Evolving, Computerizing, Personalizing. Frederick Kilgour. (Not Available EDRS)

XXII. The Shared Cataloging System of the Ohio College Library Center. Frederick G. Kilgour, Philip L. Long, Alan L. Landgraf, and John W. Wyckoff. (Not Available EDRS)

XXIII. Cataloging with a Computer - OCLC Comes to Pennsylvania. Robert C. Stewart. (Not Available EDRS)

XXIV. The Ohio College Library Center Program/Subroutine Documentation; Master Data Base Update (MDBUPD). (LI 004 425)

XXV. The Ohio College Library Center Program/Subroutine Documentation; Convert Call Number (CNVT). (LI 004 426)

XXVI. The Ohio College Library Center Program/Subroutine Documentation; Generate Pack Definition Tables (GENPDTS). (LI 004 427)

XXVII. The Ohio College Library Center Program/Subroutine Documentation; Catalog Card Format Program (CCFP). (LI 004 427)

XXVIII. The Ohio College Library Center Program/Subroutine Documentation; Build Print Tape (BPT). (LI 004 428)

XXVI

THE OHIO COLLEGE LIBRARY CENTER

PROGRAM/SUBROUTINE DOCUMENTATION

Generate Pack Definition Tables

(GENPDTS)

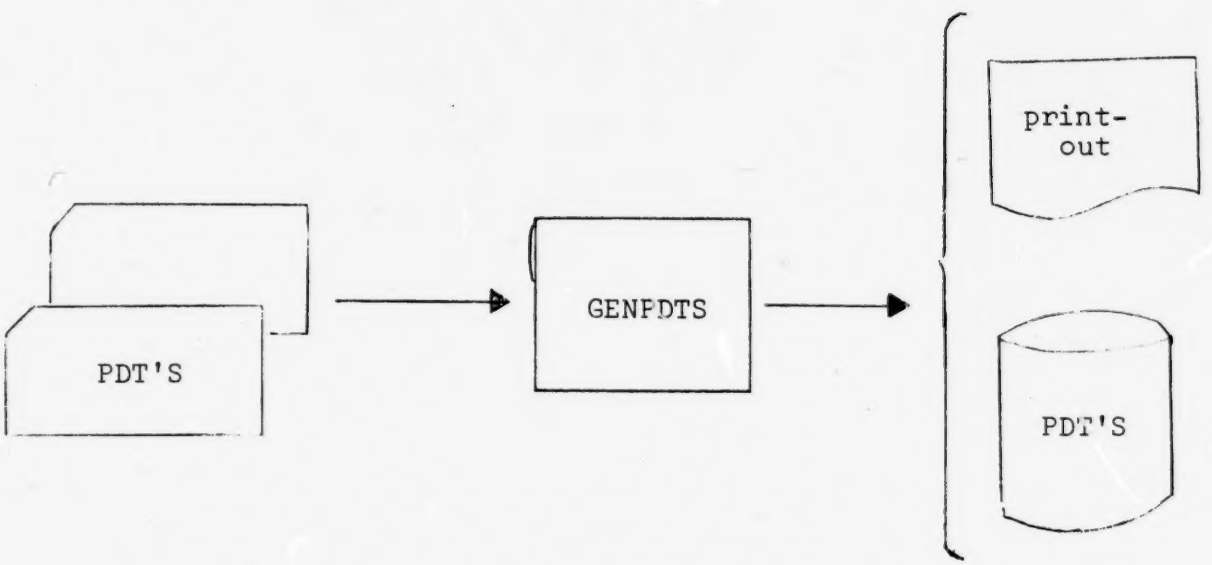
CONTENTS

		Section/Page
I.	OVERVIEW	I
II.	DATA FLOW	II
III.	SUMMARY OF INPUT AND OUTPUT	III
IV.	FUNCTIONS	IV
V.	GENERAL INFORMATION FLOW	V
VI.	SOFTWARE INTERFACE	VI
VII.	DESCRIPTION OF SPECIAL STORAGE AREAS, SWITCHES, AND TABLES	VII
	A. Special Storage	VII.1
	B. Special Switches	VII.1
	C. Special Tables	VII.2
VIII.	APPENDIX	
	A. Operating Requirements	VIII.1
	B. Operating Characteristics	VIII.3
	C. Detailed Description of Subroutines	VIII.7

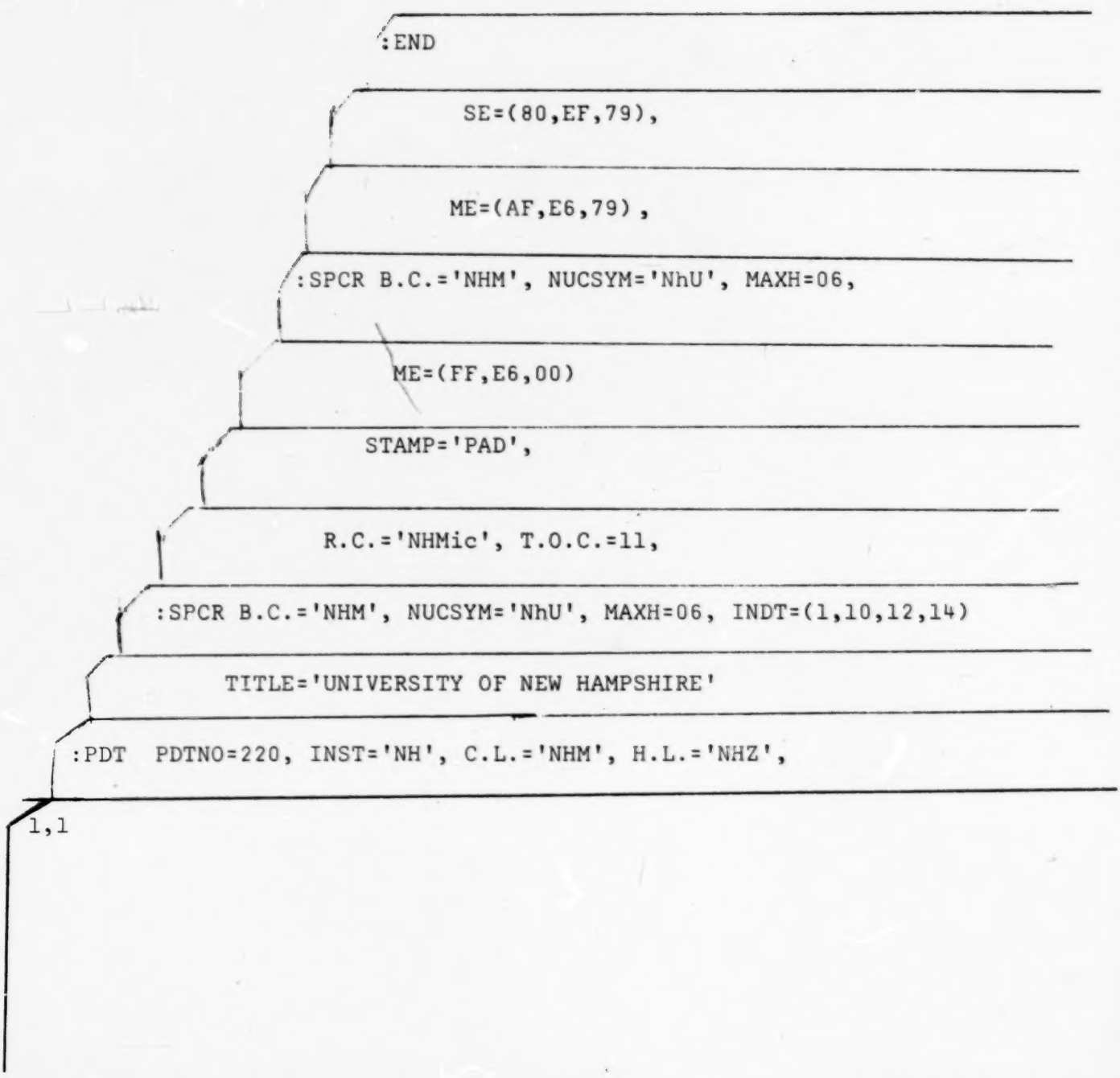
I. OVERVIEW

The FORTRAN program GENPDTS (generate the Pack Definition Tables) is used to record the institutions coded cataloging requirements (PDT'S) on disk in the format required by the program CCFP (Catalog Card Formatting Program). GENPDTS takes the general information and the specific options from card input for each receiving catalog or 'pack' in each holding library within every institution and arranges it in the required format. GENPDTS then produces an output listing for each holding library in a format similar to the original work-form. GENPDTS may be run during on-line processing.

II. DATA FLOW



III. EXAMPLE OF INPUT:



IBM DIAGRAMMING AND CHARTING WORKSHEET

Application PDT Record Date 1/25/73 Page 1 of 1

Procedure GET PDTS Output Drawn By T.B.

fixed length leader

0	PDT NO.	NO. OF SPCR's	INST. CODE	CAT. LIB. CODE
4	HOLDING LIB. CODE	ZERO	ZERO	ZERO
8	Pack I.D. Symbol	Billing Code	Location of N.U.C. Symbol	Location of Stamp
12	Indention	S.O.	MARK	MAXHL	0	SL options	ME options	TIL options
16	TIL (cont.)	SEP options	SEC options	SEM options	SET options	SUP options
20	SUC options	SUM options	SUU options	SUG options	SUT options	AEP
24	AEP (cont.)	AEC options	AEM options	AEU options	AED options	SAP options
28	SAP (cont.)	SAC options	SAM options	SAT options	Ø	ZERO
32	ZERO	ZERO

variable length trailer

Len.	N.U.C Symbol	Len.	STAMP

Fold under at dotted line.

the ohio college library center
1314 kinnear rd. - columbus ohio - 43212

Record Layout

File Name CCFP Profile Definition Tables

Record Name PDT

Record Type - ( ) Card ( ) Tape (x) Disk ( ) Other

File Organization Random Record Size 1024 Block Size 1024

General Description The Profile Definition Tables are used by CCFP in formatting text and headings on the catalog card.

FIELD NAME AND DESCRIPTION	FIELD POSITION	FIELD LENGTH	FORMAT
PDT number	1 - 4	4	binary
Number of receiving catalogs(SPCRS)	5 - 8	4	binary
Institution code	9 - 12	4	EBCDIC(right justified, trailing blanks.
Cataloging library code	13- 16	4	" " "
Holding library code	17- 20	4	" " "
(unused)	21- 24	4	X'00000000'
Zeros from this point in the record on have been compressed. A string of zeros is indicated by a X'00' followed by a binary number indicating the number of zeros to be inserted.	25- 25 26- 26	1 1	X'00' (zeros follow) X'08' (insert 8 zeros)
First receiving catalog (SPCR)	27-	variable	variable
Pack I. D.	1 - 5	5	EBCDIC
Billing code	6 - 8	3	EBCDIC
NUC symbol displacement from beginning of record(NS)	9- 10	2	binary
PDT Stamp displacement from beginning of record	11- 12	2	binary
Indentions 1,2,3,4	13- 16	4	binary
Sort order code	17- 17	1	binary

Record Layout (Cont)

File Name CCFP Profile Definition Tables

Record Name PDT

FIELD NAME AND DESCRIPTION	FIELD		
	POSITION	LENGTH	FORMAT
First receiving catalog (cont)			
Dropped heading mark	18- 18	1	EBCDIC
Maximum number of top-of-card headings	19- 19	1	binary
(unused)	20- 20	1	X'00'
22 3-byte card options (zeros compressed)	21- 86	variable	binary
Next receiving catalog (SPCR)	87- n	variable	variable
NUC symbol length (NL)	NS	1	binary
NUC sumbol	NS+1	variable	EBCDIC
PDT Stamp length	NS+1+NL	1	binary
PDT Stamp	NS+NL+2	variable	EBCDIC

IV. FUNCTIONS:

GENPDTS initially reads the two integers on the first card which indicate the value of the PRINTSW and DUMPSW. If PRINTSW = 1, an output listing is requested, otherwise no listing. If DUMPSW = 1, a hexadecimal 'dump' of the PDT is requested. The remainder of the input cards are read using two single dimensioned arrays. All the cards contained between two ':' control cards are moved to array B with the end ':' card remaining in array A.

There will be one ':PDT' and one ':END' card for every holding library. There will be one ':SPCR' card for every 'pack' or receiving catalog. The first control card encountered should be the ':PDT' card. This card will contain the PDT number, the institution code, the cataloging library code, and the holding library code. These codes are moved to locations PDTNO, INST, CL, and HL. This information is then printed if a listing is requested.

The next card encountered is the 'TITLE' card. The title of the institution is printed at the top of the page.

All the cards between the ;SPCR card and the next ':' control card including the beginning ;SPCR card are moved to array 'B'. The :SPCR card contains the billing code and the N.U.C. (National Union Catalog) symbol for this receiving catalog. It will also contain the indentions and the maximum header lines if they are different from the default.

The default for the indention indicators is (1, 10, 12, 14) and the default for the maximum heading lines is '5'. The card after the :SPCR card will contain the receiving catalog code and the sort order if it is different from the default which is '0'.

Each receiving catalog or 'pack' will have a single dimension of the array SPCRE associated with it. The receiving catalog code and the billing code will be moved to the first 8 bytes of the associated SPCRE. Each pack will also have a single dimension of the array NUCS for the N.U.C. symbol and a single dimension of the array STMP for the PDT or call number stamp if it is present. If there is no stamp, the associated STMP will be 0.

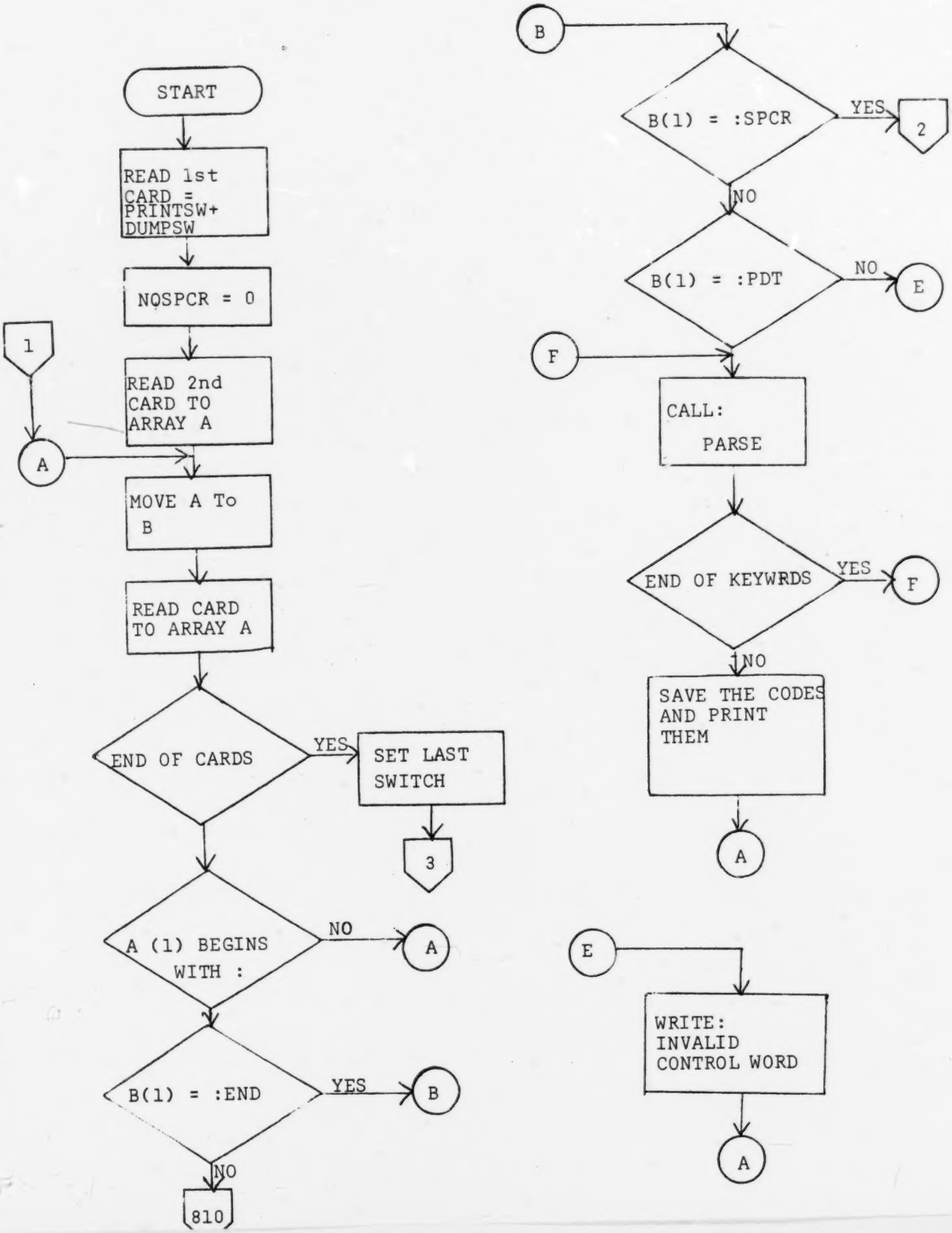
Each pack will also have a single dimension of each of the three arrays OPT1, OPT2, and OPT3 to contain all the A, B, and C options as described in the 'PDT MANUAL'. Therefore, the options for the Shelf List (sl) will be found in the first word of the associated OPT arrays. The Main Entry (me) options are found in the second word of the associated OPT arrays and so forth. If one of the group names is encountered (se, ps, etc.) the options are used for each of the group members. That is, the options found on the se card are copied into the fourth, fifth, sixth, and seventh words of the associated OPT arrays to be used as the sep, sec, sem, and set options.

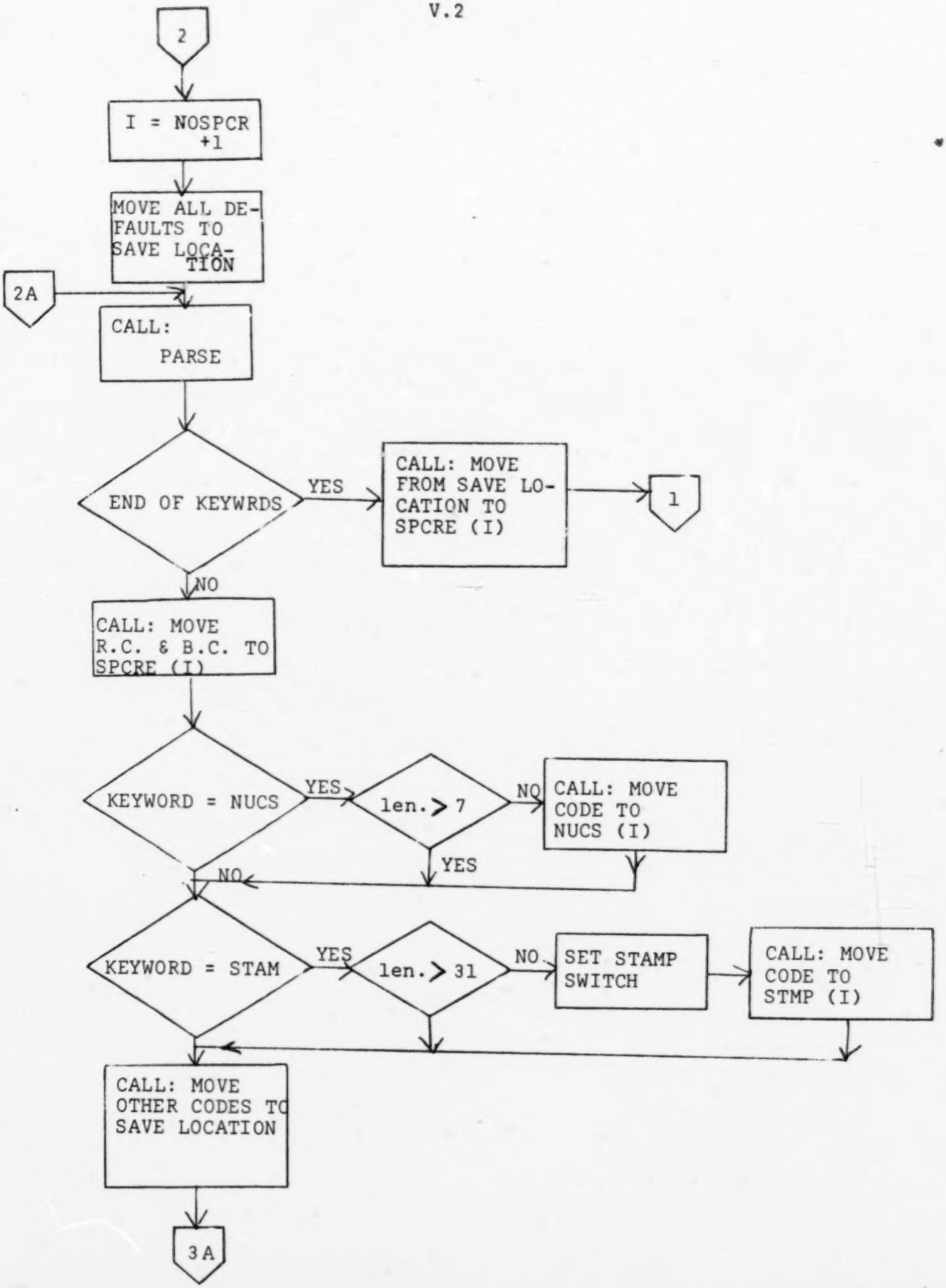
The indention indicators are moved to bytes 13 - 16 of the

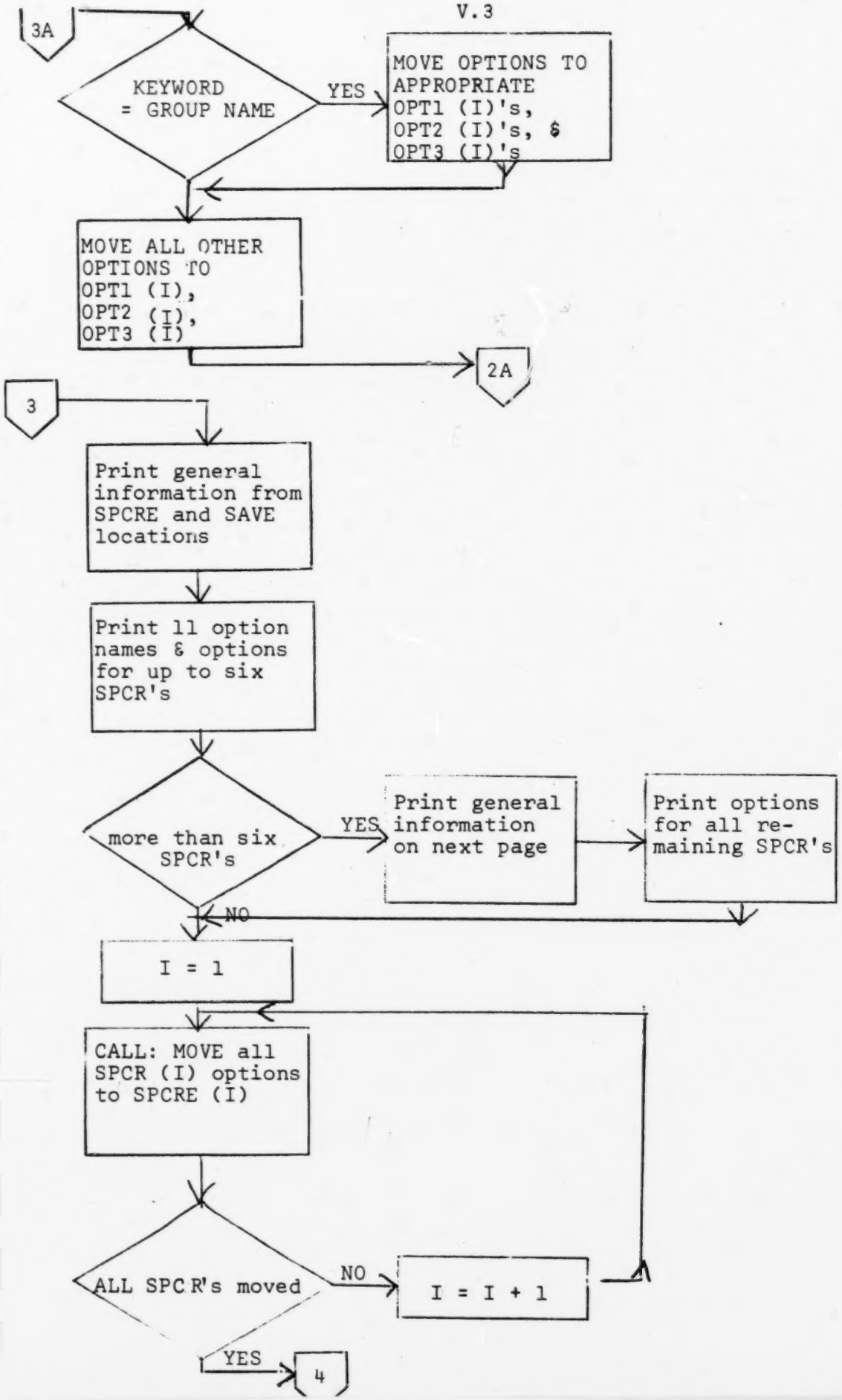
corresponding SPCRE array. The sort order, the dropped heading mark, and the maximum heading lines are moved to bytes 17 - 19 of the corresponding SPCRE array.

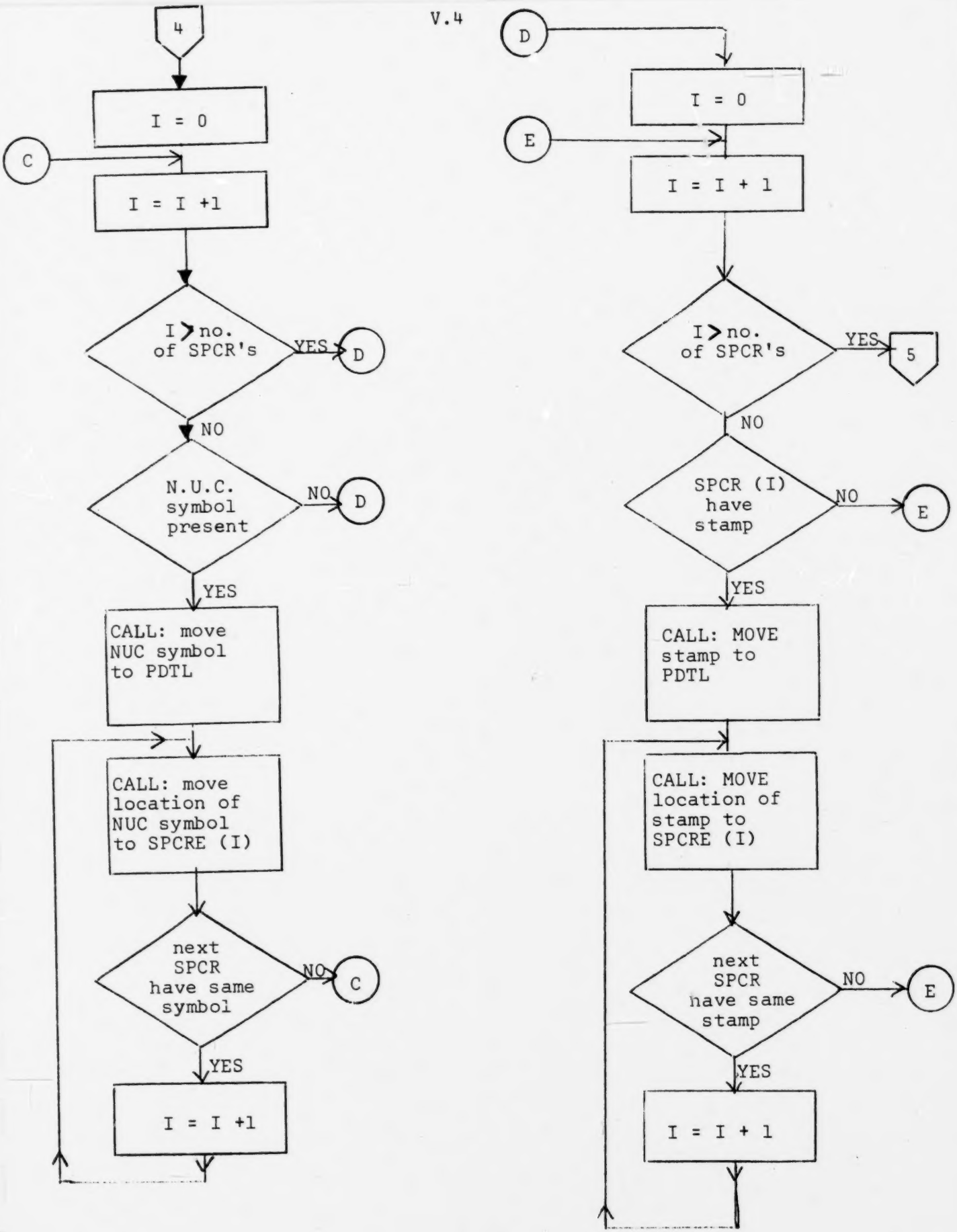
After all the :SPCR packs have been processed, the next card read should be the :END card. At this point, all the information contained in the temporary storage locations (NUCS, STMP, etc.) all the SPCRE arrays, and all the OPT arrays are printed out and a hexadecimal dump is produced if requested. All this information is then copied to a location called RECORD and a CAL3,0 is issued to write the contents of RECORD onto the appropriate disk file.

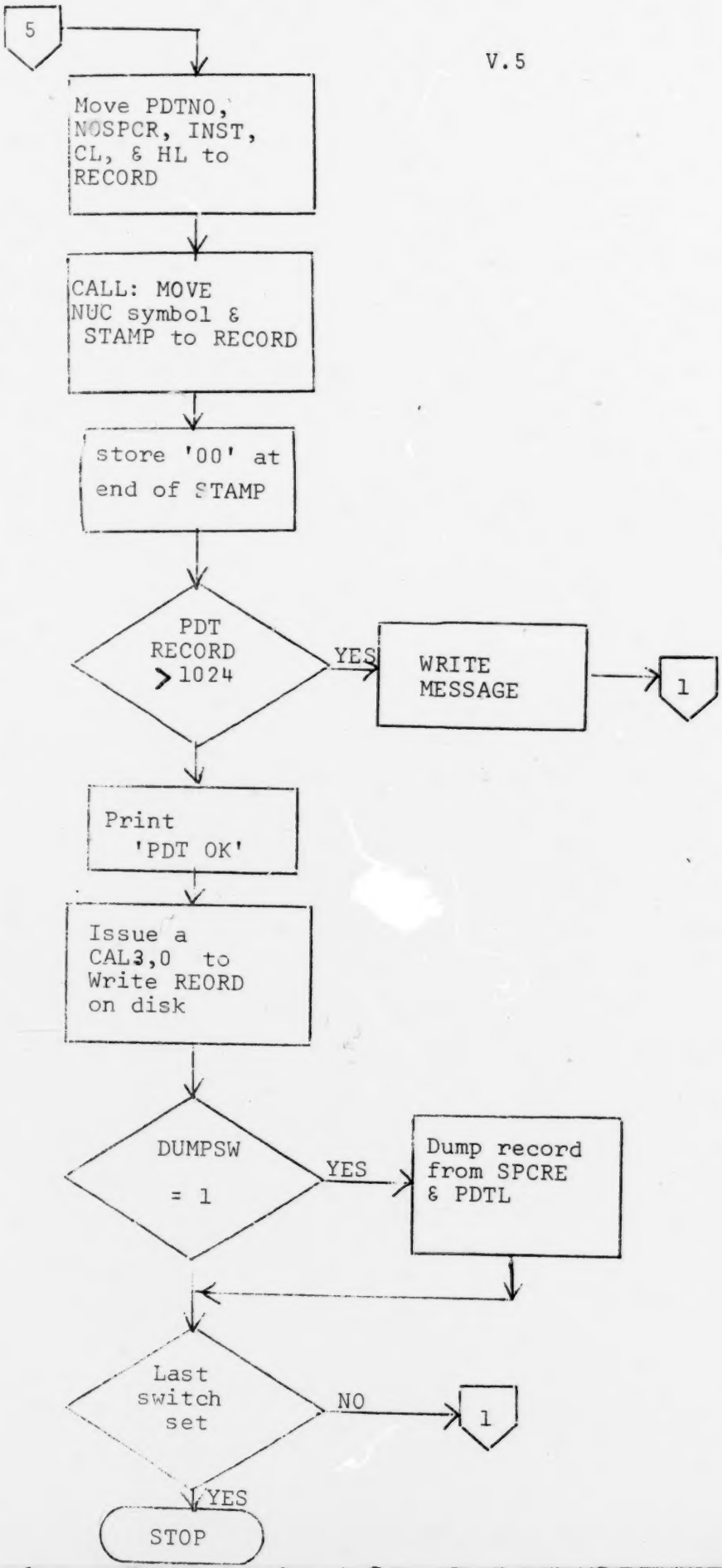
V. GENERAL FLOW CHART




2
I = NOSPCR +1
MOVE ALL DEFAULTS TO SAVE LOCATION
2A
CALL: PARSE
END OF KEYWRDS
YES
CALL: MOVE FROM SAVE LOCATION TO SPCRE (I)
1
NO
CALL: MOVE R.C. & B.C. TO SPCRE (I)
KEYWORD = NUCS
YES
len. > 7
NO
CALL: MOVE CODE TO NUCS (I)
YES
NO
KEYWORD = STAM
YES
len. > 31
NO
SET STAMP SWITCH
CALL: MOVE CODE TO STMP (I)
CALL: MOVE OTHER CODES TO SAVE LOCATION
3A


3A
KEYWORD = GROUP NAME
YES
MOVE OPTIONS TO APPROPRIATE OPT1 (I)'s, OPT2 (I)'s, & OPT3 (I)'s
MOVE ALL OTHER OPTIONS TO OPT1 (I), OPT2 (I), OPT3 (I)
2A
3
Print general information from SPCRE and SAVE locations
Print 11 option names & options for up to six SPCR's
more than six SPCR's
YES
Print general information on next page
Print options for all remaining SPCR's
NO
I = 1
CALL: MOVE all SPCR (I) options to SPCRE (I)
ALL SPCR's moved
NO
I = I + 1
YES
4


4
I = 0
C
I = I +1
I > no. of SPCR's
YES
D
NO
N.U.C. symbol present
NO
D
YES
CALL: move NUC symbol to PDTL
CALL: move location of NUC symbol to SPCRE (I)
next SPCR have same symbol
NO
C
YES
I = I +1
D
I = 0
E
I = I + 1
I > no. of SPCR's
YES
5
NO
SPCR (I) have stamp
NO
E
YES
CALL: MOVE stamp to PDTL
CALL: MOVE location of stamp to SPCRE (I)
next SPCR have same stamp
NO
E
YES
I = I + 1


5
Move PDTNO, NOSPCR, INST, CL, & HL to RECORD
CALL: MOVE NUC symbol & STAMP to RECORD
store '00' at end of STAMP
PDT RECORD > 1024
YES
WRITE MESSAGE
1
Print 'PDT OK'
Issue a CAL3,0 to Write REORD on disk
DUMPSW = 1
YES
Dump record from SPCRE & PDTL
Last switch set
NO
1
YES
STOP

VI. SOFTWARE INTERFACE

A. Linkage - Background Linkage with OBM

B. Parameter List - none

C. Return Codes - If GENPDTS was able to process all the cards for a holding library, the message 'PDT OK' is printed.

D. Other Entry Points - none

E. OCLC External Subroutines Referenced:

MOVE
PARSE

F. OCLC Procedures Referenced - none

VII. SPECIAL STORAGE AREAS, SWITCHES, AND TABLES

A. Special Storage Areas

PDTL - This is a nine-word storage area that is used to store the general information for one holding library. This includes the PDT number, number of SPCR's, institution code, cataloging library code, holding library code, and the receiving catalog code.

SPCRE - This is a two dimensioned array. One 25-word storage area of SPCRE is associated with each of the SPCR's up to a maximum of 20 SPCR's. The associated SPCRE will contain the general information for that SPCR. This includes the receiving catalog code, billing code, indention indicators, sort order, dropped heading mark, and the maximum number of heading lines.

OPT1, OPT2, OPT3 - Each of these is a two dimensioned array. One 26-word storage area in each array is associated with each of the SPCR's up to a maximum of 20 SPCR's. The area in OPT1 will contain the 22 'A' options; OPT2 will contain the 22 'B' options; and OPT3 will contain the 22 'C' options for one SPCR or 'pack'.

NUCS - This is a two dimensioned array. One 2-word storage area of NUCS is associated with each of the SPCR's up to a maximum of 20 SPCR's. The associated area in NUCS will contain the N.U.C. symbol for that SPCR or 'pack'.

STMP - This is a two-dimensioned array. One 8-word storage area of STMP is associated with each of the SPCR's up to a maximum of 20 SPCR's. The associated area in STMP will contain the 'PDT' or call number stamp for that SPCR or 'pack'.

B. Special Switches:

LAST - This switch is set to '0' until the last card is read, at which time LAST is set to '1'.

PDTOK - This switch is used to indicate to the program whether or not the information for a particular PDT number or holding library was correct and able to be formatted. If PDTOK is set to '1' after all processing for a holding library is complete, the message 'PDT OK' is printed at the bottom of the listing. PDTOK is set to '0' when something has gone wrong with the processing for that holding library, and no message is printed.

SPCROK - This switch operates much the same way as the PDTOK switch. If there was nothing wrong with the options for any SPCR or pack, the switch SPCROK is set to '1' and the 'PDT OK' message is printed.

SW - This switch is used to indicate whether or not the institution code, the cataloging library code, and the holding library code are present on the ':PDT' card. If any one or all of the codes are missing, the switch 'SW' is set to 0 and the message 'MISSING LIBRARY IDENTIFIER' is printed. If all three codes are present, the 'SW' switch is set to '1'.

The 'SW' switch is also used in the same way to test for the presence of the receiving catalog code, the billing code, the N.U.C. symbol, and at lease one card option for each SPCR or 'pack'.

PRINTSW - This switch is used to indicate to the program whether or not a listing of the card input is requested. If PRINTSW is set to '1', a listing is requested. If PRINTSW is set to '0', no listing is required.

DUMPSW - This switch is used to indicate to the program whether or not a hexadecimal 'dump' of the card input is requested. If DUMPSW is set to '1', a dump is requested. If DUMPSW is set to '0', no listing is required.

C. Special Tables

KEYWD1 - This is a table of the keyword codes which are found on the :PDT card.

KEYWD2 - This is a table of the possible keyword codes which could be found on :SPCR card.

KEYWD3 - This is a table of the keyword codes that are used as group names for the specific options. Also included in this table are the keyword codes for the single entry 'groups', such as 'sl' and 'me'.

KEYWD4 - This is a table of the keyword codes for the specific options within the groups.

I1 - This is a table of indexes to the group names in the KEYWRD4 table.

I2 - This is a table of indexes which point to the first item name of the group in the KEYWD4 table.

I3 - This is a table which contains the number of items in each group.

I4 - This is a table of displacements which point to where the start of each group is to be in the appropriate 'OPT' array.

Examples of the use of the 'I' tables:

The entries for each 'I' table for the ATS group are respectively : 10, 16, 3, 19. These mean that ATS is the 10th group name in the KEYWD3 table. The first item in the ATS group is the 16th entry in the KEYWD4 table. There are three items in the ATS group. The start of the group is to be the 19th entry in the appropriate 'OPT' array.

APPENDIX A

OPERATING REQUIREMENTS

1. Computer - Xerox Sigma 5

2. I/O Devices - 1 9-track tape drive, line printer, card reader.

3. Operating System - OBM

4. Execution Time - depends on number of PDT's to be run -- less than one minute wall clock time per PDT.

5. Run Schedule - Whenever new PDT is added or existing PDT is updated.

6. Job Control Language

```
!JOB   RUN,GENPDTS
!ATT
!PAU  -S4C
!OLOAD  (MAP,PROG)
:ROOT  (DEV,CRA03,EOD)
      \
       \      Binary deck of GENPDTS
       /
      /
!EOD
!ROV
1,1       PRINT & DUMP
   \
    \     Control Cards for PDTS
    /
   /
!EOD
!FIN
```

To make updates to existing PDTS, the 'PDT' tape is required. The first file on this tape contains the update cards. The second file contains the program GENPDTS. The third file contains the existing PDT records. Job Control cards required:

```
!JOB   UPDATE,PDT's
!ATT
!ALLOBT  (FILE,GO), (FSIZE,0),SAVE
!ALLOBT  (FILE,OV), (FSIZE,35),SAVE
!ALLOBT  (FILE,x1),FSIZE,25000), (FOR,C), SAVE
!PAU    --MOUNT INPUT TAPE ON T0
!SFIL  T0
!ASSIGN  (M:BI,T0)
!ASSIGN  (M:BO,BT,OV)
!COPY  BI,1
```

```
!ASSIGN  (M:CI, T0)
!MACRSYM  SI,CI
+1,1       PRINT & DUMP.
   (
   )       PDT updates with update cards
   (
+END
!IMG002A  E1
!ASSIGN  (F:105,x1)
!ROV
!UNL T0
!FIN
```

APPENDIX B

OPERATING CHARACTERISTICS

1. Console Messages

MESSAGE:

!PAU --MOUNT PDT TAPE ON T0

RESPONSE:

MOUNT Tape as directed
'C'
NEW LINE

MESSAGE:

PAUSE -- PERM DISK WRITE ERROR

RESPONSE:

'X' -- abort the job
'C' -- continue to process the next card

2. PROGRAMMED ABNORMAL COMPLETION -

GENPDTS is not programmed to terminate abnormally

3. DIAGNOSTICS - ERRORS

GENPDTS will reject a card and take a direct branch to read the next ':' control card under the following conditions

MESSAGE	REASON
***PDT NUMBER = ' '. INVALID STATEMENT FOLLOWS	Switch 'SW' is set to 0 -- see description of 'SW' on page
***SYMTAX ERROR	Three Reasons: 1. A key word on the ;PDT or :SPCR card ends with a ',' instead of a '='. 2. A keyword does not end with '='. 3. Options were not enclosed in parenthesis '( )'.
**INVALID KEYWORD	A keyword is not in the list of valid keywords.
***MISSING PDT NUMBER	The PDTNO keyword on the :PDT card is equal to 0.

MESSAGE	REASON
***RECEIVING CATALOG CODE MISSING	self-explanatory
***BILLING CODE MISSING	o o o
***NUC SYMBOL MISSING	o o o
***NO CARD OPTIONS	There must be at least one option card for each SPCR.
***NUC SYMBOL TOO LARGE	The NUC symbol cannot be more than seven characters long
*** PDT STAMP TOO LARGE	The PDT stamp cannot be greater than 31 characters long
***PDT RECORD TOO LARGE	The entire PDT record cannot be larger than 1024 bytes long

5. PARAMETER CARDS REQUIRED - first card of input must contain the PRINTSW and DUMPSW in columns 1 and 2.

6. EXAMPLE OF OUTPUT - Next Page

EXAMPLE OF OUTPUT LISTING

PDT NUMBER = 891 BOSTON UNIVERSITY--AFRICAN STUDIES
INSTITUTION ..... BO
CATALOGING LIBRARY ... BOS
HOLDING LIBRARY ..... BOV

PACK ID SYMBOL	BOSZC	BOSXC	BOSAT	BOSSC	BOVSL	BOVME
BILLING CODE	BOS	BOS	BOS	BOS	BOS	BOS
NUC SYMBOL	MBU	MBU	MBU	MBU	MBU	MBU
PDT STAMP	AFR. ST.	AFR. ST.	AFR. ST.	AFR. ST.	AFR. ST.	AFR. ST.
INDENTIONS	1 10 12 14	1 10 12 14	1 10 12 14	1 10 12 14	1 10 12 14	1 10 12 14
SORT ORDER CODE	11	21	00	00	42	41
DROP. HD. MARK	>	>	>	>	>	>
MAX HD. LINES	8	8	8	8	8	8
SPECIFIC OPTIONS						
SL	00 00 00	00 00 00	00 00 00	00 00 00	8F E6 00	00 00 00
ME	FF E6 00	8F 66 00	BF E6 00	00 00 00	00 00 00	8F E6 00
TIL	00 00 00	00 00 00	8F 67 08	00 00 00	00 00 00	00 00 00
SE						
SEP	00 00 00	00 00 00	8F 6F 6E	00 00 00	00 00 00	00 00 00
SEC	00 00 00	00 00 00	8F 6F 6E	00 00 00	00 00 00	00 00 00
SEM	00 00 00	00 00 00	8F 6F 6E	00 00 00	00 00 00	00 00 00
SET	00 00 00	00 00 00	8F 68 08	00 00 00	00 00 00	00 00 00
PS						
SUP	00 00 00	00 00 00	00 00 00	8F 6E 6E	00 00 00	00 00 00
SUC	00 00 00	00 00 00	00 00 00	8F 6E 6E	00 00 00	00 00 00
SUM	00 00 00	00 00 00	00 00 00	8F 6E 6E	00 00 00	00 00 00
GS						
SUU	00 00 00	00 00 00	00 00 00	8F 6E 6E	00 00 00	00 00 00
SUG	00 00 00	00 00 00	00 00 00	8F 6A 08	00 00 00	00 00 00
TS						
SUT	00 00 00	00 00 00	00 00 00	8F 6A 08	00 00 00	00 00 00
PAE						
AEP	00 00 00	00 00 00	8F 6F 6E	00 00 00	00 00 00	00 00 00
AEC	00 00 00	00 00 00	8F 6F 6E	00 00 00	00 00 00	00 00 00
AEM	00 00 00	00 00 00	8F 6F 6E	00 00 00	00 00 00	00 00 00
TAE						
AEU	00 00 00	00 00 00	8F 6F 6E	00 00 00	00 00 00	00 00 00
AED	00 00 00	00 00 00	8F 68 08	00 00 00	00 00 00	00 00 00
ATS						
SAP	00 00 00	00 00 00	8F 6F 6E	00 00 00	00 00 00	00 00 00
SAC	00 00 00	00 00 00	8F 6F 6E	00 00 00	00 00 00	00 00 00
SAM	00 00 00	00 00 00	8F 6F 6E	00 00 00	00 00 00	00 00 00
TTS						
SAT	00 00 00	00 00 00	8F 68 08	00 00 00	00 00 00	00 00 00

* PDT OK *

EXAMPLE OF OUTPUT 'DUMP'

```
00003378  00000006  C2D64040  C2D6E240  C2D6E540  00000000  00000000  00000000

C2D6E2A9  83C2D6E2  027E0027  010A0C0E  116E0800  000000FF  E6000000  00000000  00000000  00000000  00000000  00000000
00000000  00000000  00000000  00000000  00000000  00000000  00000000  00000000  00000000  00000000  00000000  00000000

C2D6E2A7  83C2D6E2  027E0027  010A0C0E  216E0800  0000006F  066D0000  00000000  00000000  00000000  00000000  00000000
00000000  00000000  00000000  00000000  00000000  00000000  00000000  00000000  00000000  00000000  00000000  00000000

C2D6E281  A3C2D6E2  027E0027  010A0C0E  006E0800  0000008F  E6008F67  0B8F6F6E  8F6F6E3F  6F6E3F68  0B000000  00000000
00000000  00000000  0000003F  6F6E3F6F  6E3F6F6E  8F6F6E3F  6B0B8F6F  6E3F6F6E  8F6F6E3F  6B0B0000  00000000  00000000  00000000

C2D6E2A2  83C2D6E2  027E0027  010A0C0E  006E0800  00000000  00000000  00000000  00000000  00000000  00000000  008F6E6E  8F6E6E8F
6E6E3F6E  6E3F6A08  3F6A3B00  00000003  00000000  00000000  00000000  00000000  00000000  00000000  00000000  00000000  00000000

C2D6E5A2  93C2D6E2  027E0027  010A0C0E  426E0800  8FE60000  00000000  00000000  00000000  00000000  00000000  00000000
00000000  00000000  00000000  00000000  00000000  00000000  00000000  00000000  00000000  00000000  00000000  00000000  00000000

C2D6E594  85C2D6E2  027E0027  010A0C0E  416E0800  0000008F  E6000000  00000000  00000000  00000000  00000000  00000000
00000000  00000000  00000000  00000000  00000000  00000000  00000000  00000000  00000000  00000000  00000000  00000000  00000000

03D4C2E4  08C18699  4840E2A3
```

APPENDIX C

SUBROUTINE DOCUMENTATION

I. FUNCTIONS

The purpose of PARSE is to break down the keywords and codes on the :PDT and :SPCR cards. The calling routine provides a location to contain the one word keyword and another location to contain the returned parameter. The returned parameter may be one of three types. If it is a type '0', the parameter is one word long and is right justified. If it is a type '1', a single parameter that was enclosed in quotation marks is returned without the quotes. If it is a type '2', one or more parameters that were enclosed in parentheses are returned without the parentheses. Type 1 and Type 2 parameters are left justified. PARSE is an assembler subroutine which may be called by the FORTRAN program GENPDTS.

II. SOFTWARE INTERFACE

A. Linkage

The FORTRAN calling swquence for PARSE is:

CALL PARSE (B(1), POS, LEN1, KEYWD, C(1), LENZ, TYPE, 300,900)
where the parameters are described below.

This FORTRAN call is equivalent to the assembler calling sequences:

```
LI,R14          1
BAL,R15         PARSE
ARG             B(1)
ARG             POS
ARG             LEN1
ARG             KEYWD
ARG             C(1)
ARG             LEN2
ARG             TYPE
ARG             300
ARG             900
```

B. Parameter List Description

B(1)	--	beginning of the source text string
POS	--	byte displacement within 'B' to the first byte that is to be parsed
LEN1	--	maximum length of 'B' in bytes
KEYWD	--	word to contain the returned keyword
C(1)	--	array to contain the returned parm(s)
LEN2	--	length in bytes of the returned parm field
TYPE	--	either 0,1, or 2 as described in 'FUNCTIONS'
300	--	exit taken if no more keywords or parms
900	--	exit taken if syntax error is detected

C. Return Codes - none

D. Other Entry Points - none

E. OCLC Subroutines Referenced - none

F. OCLC Procedures Referenced - none

I. FUNCTIONS

This routine is used to move byte strings from one location to another. MOVE is the entry point for the FORTRAN call; MOVEBYTs is the entry point for the assembler call.

II. SOFTWARE INERFACE

A. Linkage

The FORTRAN calling sequence for MOVE is:

```
CALL MOVE (A(1), AOFF, B(1), BOFF, NBYTES)
```

Where AOFF and BOFF are optional.

The assembler calling sequence for MOVEBYTS is:

```
LI,R2     BA(SOURCE)
LI,R3     BA(TARGET)
LI,R4     BYTE COUNT
BAL,R1    MOVEBYTES
```

B. Parameter List Description

A(1)	--	beginning of the source text string
AOFF	--	byte displacement within 'A' to the first byte that is to be moved
B(1)	--	beginning of the target array
BOFF	--	byte displacement within 'B' to the first byte location
NBYTES	--	number of bytes to be moved

C. Return Codes - none

D. Other Entry Points - MOVEBYTES

E. OCLC Subroutines Referenced - none

F. OCLC Procedures Referenced - none

ED 080122

XXVII

THE OHIO COLLEGE LIBRARY CENTER
PROGRAM/SUBROUTINE DOCUMENTATION

Catalog Card Format Program
(CCFP)

CONTENTS

		Section/Page
I.	OVERVIEW	I
II.	DATA FLOW	II
III.	SUMMARY OF INPUT AND OUTPUT	III
	CNVT Output	III.2
	CCFP Output	III.5
	CCFP PDT'S	III.7
IV.	FUNCTIONS	IV
V.	GENERAL INFORMATION FLOW	V
VI.	SOFTWARE INTERFACE	VI
VII.	DESCRIPTION OF SPECIAL STORAGE AREAS, SWITCHES, AND TABLES	VII
	A. Special Storage	
	PDT Buffer	VII.1
	Seppacked Paragraph	VII.2
	B. Switches	VII.2
	C. Tables	VII.3
VIII.	APPENDIX	VIII
	A. Operating Requirements	VIII.2
	B. Operating Characteristics	VIII.3
	C. Detailed Description of Internal Subroutines	VIII.7
	D. Additional Procedure Documentation	VIII.79
	F. Internal Subroutine Cross-Reference	VIII.99

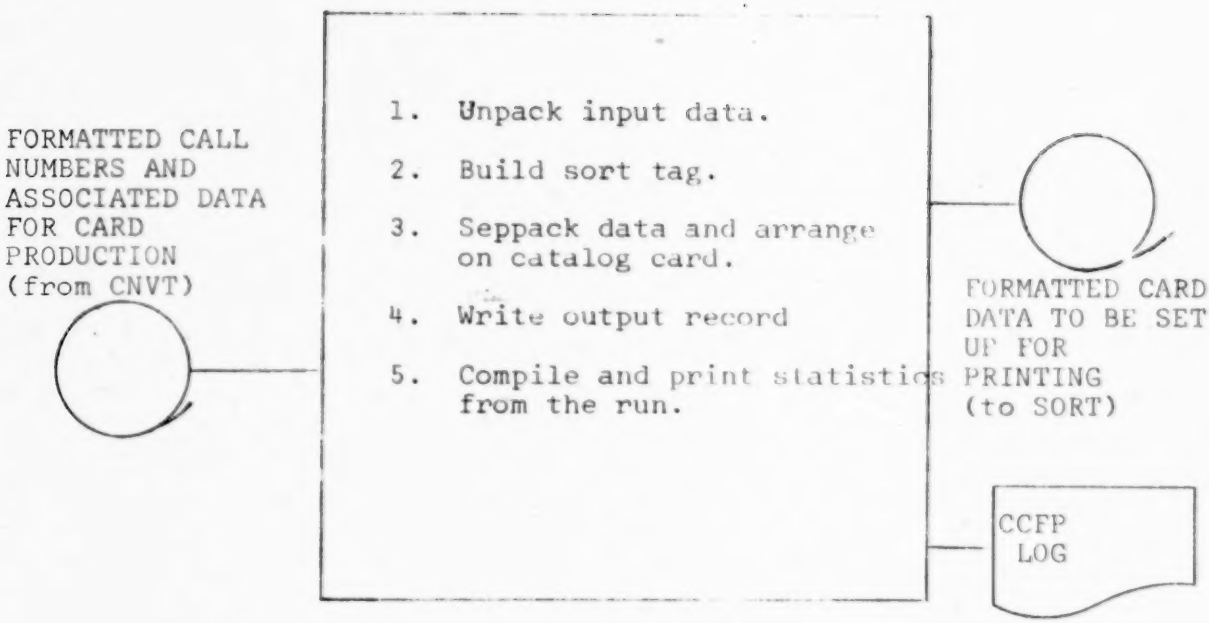
OCLC PROGRAM DOCUMENTATION

I. OVERVIEW

CCFP is the second step in the formatting and production of catalog cards to be sent to Members. It has as input the formatted call number and associated data for card production from CNVT. CCFP formats the text paragraphs from the input record and builds headings. It then arranges the call number, the headings, and the text as they will appear on the catalog card. Depending on pre-determined specifications for each Member library, CCFP creates a card for each catalog set up by the Member, varying the headings and setting indentions as prescribed by the Member.

The output of CCFP is records on tape which contain formatted paragraphs, and sort and control information to be used in setting up the cards for printing.

II. DATA FLOW


FORMATTED CALL NUMBERS AND ASSOCIATED DATA FOR CARD PRODUCTION (from CNVT)
1. Unpack input data.
2. Build sort tag.
3. Seppack data and arrange on catalog card.
4. Write output record
5. Compile and print statistics from the run.
FORMATTED CARD DATA TO BE SET UP FOR PRINTING (to SORT)
CCFP LOG

III. SUMMARY OF INPUT AND OUTPUT

the ohio college library center
1314 kinnear rd. - columbus ohio - 43212

Record Layout

File Name CNVT OUTPUT RECORD

Record Name _____

Record Type - ( ) Card (X) Tape ( ) Disk ( ) Other _____

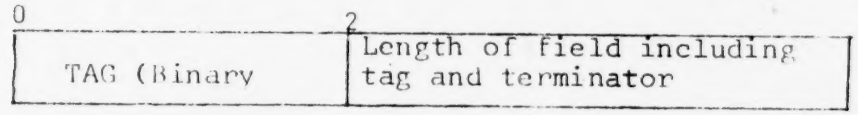
File Organization SEQUENTIAL Record Size 4096 Block Size UNBLOCKED

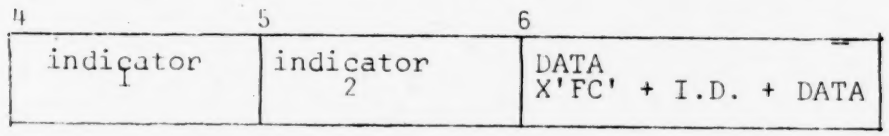
General Description _____

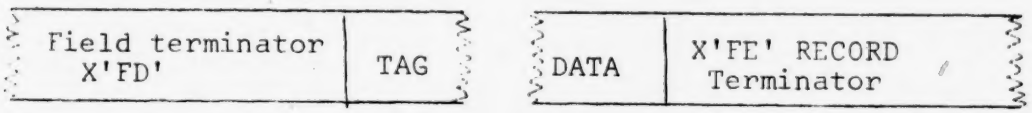
FIELD NAME AND DESCRIPTION	FIELD POSITION	FIELD LENGTH	FIELD FORMAT
RECORD LEADER			
PDT Number	0-1	2	Binary
Unit Card Indicator	2 (bit 0)	1 bit	Binary
Added Copy Indicator	2(bit 1)	1 bit	Binary
Reserved	2(bits 2-7)	6 bits	
Number of Extra Cards	3	1	Binary
OCLC Number	4-7	4	Binary
LC Card Number	8-19	12	EBCDIC
Reserved	20-27	8	Binary
Language Code Index	28-29	2	Binary

Following the leader are the tag fields. These fields are variable in length and immediately follow the previous field (or leader for the first element). Each is terminated by a delimiter (S'FD' or X'FE' for the last element). They follow the basic MARC II format with the following exceptions:

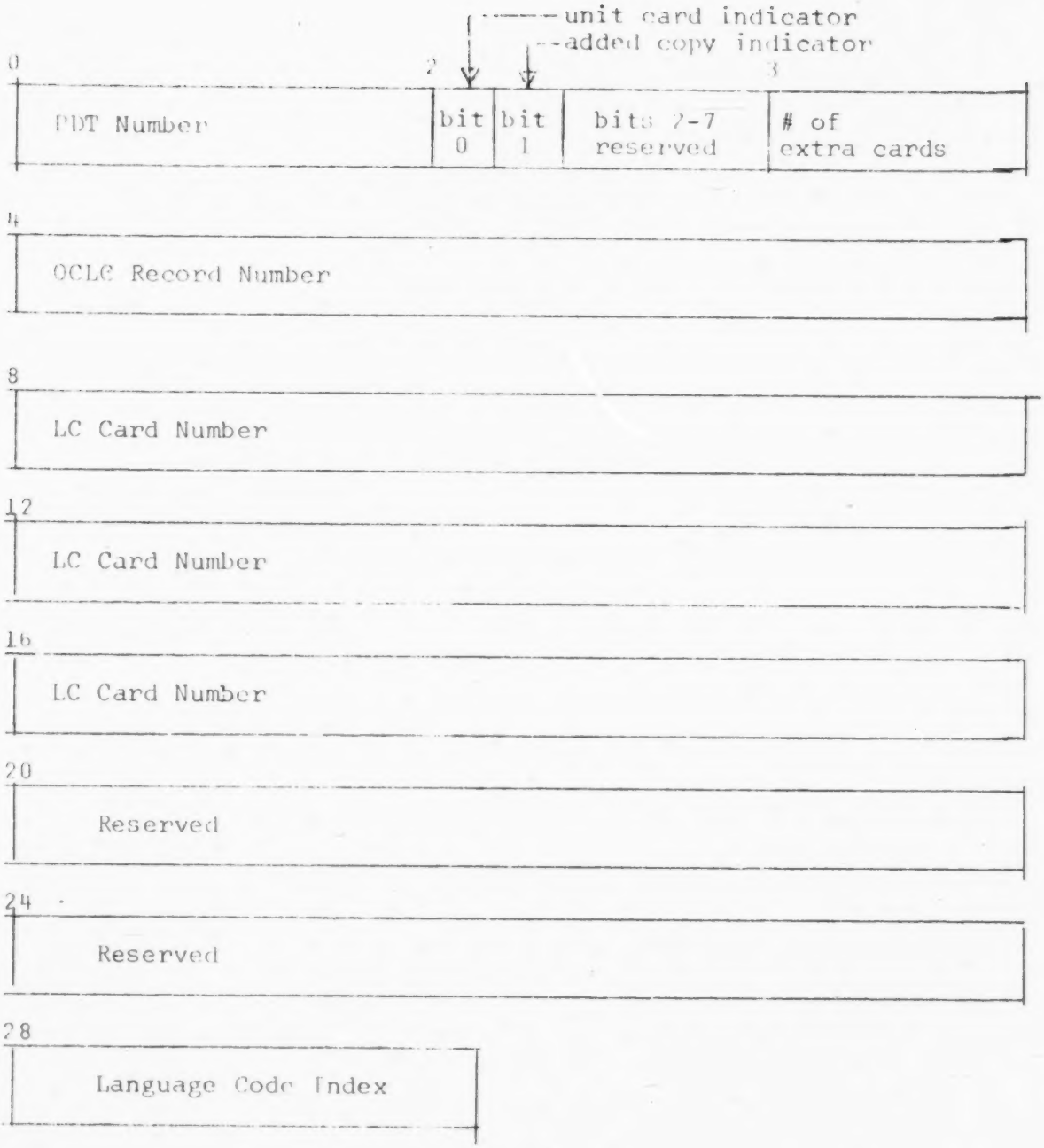
1. The tag itself is Binary
2. The field length is Binary
3. ‡P (X'PC97'), ‡B (X'FC82'), or ‡C (X'FC83') within the call number field are used to indicate that a stamp is to be placed in that position relative to the call number.







CNVT OUTPUT


unit card indicator
added copy indicator
0
2
3
PDT Number
bit 0
bit 1
bits 2-7 reserved
# of extra cards
4
OCLC Record Number
8
LC Card Number
12
LC Card Number
16
LC Card Number
20
Reserved
24
Reserved
28
Language Code Index

the ohio college library center
1314 kinnear rd. - columbus ohio - 43212

Record Layout

File Name CCFP Output Records

Record Name

Record Type - ( ) Card (X) Tape ( ) Disk ( ) Other

File Organization Sequential Record Size 1200 Block Size 1200

General Description CCFP Output Records contain formatted lines and headings and control information necessary to sort and build the catalog cards.

FIELD NAME AND DESCRIPTION	FIELD POSITION	FIELD LENGTH	FORMAT
Block size	1 - 2	2	binary
Reserved	3 - 4	2	X'0000'
Record length	5 - 6	2	binary
Reserved	7 - 8	2	X'0000'
Sort tag	9 - 80	72	variable
first tag entry	9 - 13	5	EBCDIC
second tag entry	14- 56	43	variable
third tag entry	57- 80	24	variable
Output record number	81- 82	2	binary
Logical record number (for spanned records)	83- 84	2	binary
Number of cards this record will produce	85- 85	1	binary
Number of extra cards	86- 86	1	binary
Card column for zeroth indention for call numbers	87- 87	1	binary
Card column for zeroth indention for other text lines	88- 88	1	binary
Unused	89- 90	2	X'0000'
Pointer to heading paragraph text	91- 92	2	binary
Pointer to first text paragraph	93- 94	2	binary
Pointer to continuation policy	95- 96	2	binary
Continuation line number for text on extension cards	97- 97	1	binary

Record Layout (Cont)

File Name CCFP Output Records

Record Name

FIELD NAME AND DESCRIPTION	FIELD		
	POSITION	LENGTH	FORMAT
Billing code	98 - 100	3	EBCDIC
Text paragraphs	101- n (n - 1200)	variable	variable

Each paragraph is preceded by its beginning line number and ended by a X'FF'. A X'FF' also begins the call number paragraph followed by the call number beginning line number, then the call number text. X'FD' indicates that a one-byte tab value follows, then text. X'FE' denotes the end of a line within a paragraph. X'FC' indicates that the following byte is a diacritical.

the ohio college library center
1314 kinnear rd. - columbus ohio - 43212

Record Layout

File Name CCFP Profile Definition Tables

Record Name PDT

Record Type - ( ) Card ( ) Tape (X) Disk ( ) Other

File Organization Random Record Size 1024 Block Size 1024

General Description The Profile Definition Tables are used by CCFP in formatting text and headings on the catalog card.

FIELD NAME AND DESCRIPTION	FIELD POSITION	FIELD LENGTH	FORMAT
PDT number	1 - 4	4	binary
Number of receiving catalogs(SPCRS)	5 - 8	4	binary
Institution code	9 - 12	4	EBCDIC(right justified, trailing blanks.
Cataloging library code	13- 16	4	" " "
Holding library code	17- 20	4	" " "
(unused)	21- 24	4	X'00000000'
Zeros from this point in the record on have been compressed. A string of zeros is indicated by a X'00' followed by a binary number indicating the number of zeros to be inserted.	25- 25 26- 26	1 1	X'00' (zeros follow) X'08' (insert 8 zeros)
First receiving catalog (SPCR)	27-	variable	variable
Pack I. D.	1 - 5	5	EBCDIC
Billing code	6 - 8	3	EBCDIC
NUC symbol displacement from beginning of record(NS)	9- 10	2	binary
PDT Stamp displacement from beginning of record	11- 12	2	binary
Indentions 1,2,3,4	13- 16	4	binary
Sort order code	17- 17	1	binary

Record Layout (Cont)

File Name CCFP Profile Definition Tables

Record Name PDT

FIELD NAME AND DESCRIPTION	FIELD		
	POSITION	LENGTH	FORMAT
First receiving catalog (cont)			
Dropped heading mark	18- 18	1	EBCDIC
Maximum number of top-of-card headings	19- 19	1	binary
(unused)	20- 20	1	X'00'
22 3-byte card options (zeros compressed)	21- 86	variable	binary
Next receiving catalog (SPCR)	87- n	variable	variable
NUC symbol length (NL)	NS	1	binary
NUC sumbol	NS+1	variable	EBCDIC
PDT Stamp length	NS+1+NL	1	binary
PDT Stamp	NS+NL+2	variable	EBCDIC

I. FUNCTIONS

At its entry point, CCFP saves the current time and logs its name on the TY device. It then sets up its buffers and temporary work area (temp area) in the background temp area. For every input transaction read, CCFP re-initializes its field and record pointers. If a read has not been issued, it issues a CALl,1 to read the next input record. It then picks up information from the input record leader and sets up the number of the Profile Definition Table (PDT) to be used and processing indices for this request. The routine FMT:RPDT is used to read the desired PDT and move it into the PDT buffer PDT:L.

After all necessary switches and tables are re-initialized, every field in the input record is mapped using the routine FMT:MAP. The addresses of all the packed input fields are stored in 'P:' table entries (e.g. P:ME, P:CARDNO, P:CN). Then the Library of Congress card number (LCCN) is unpacked and formatted in what will be the bottom, or authority, line of the card, Next the following input fields are unpacked in turn by branching to the routine FMT:CC: call number; main entry; uniform title or title; edition statement and imprint, if present; special notes; collation statement and series statement, if present; bibliography notes. The unpack process consists of picking up each character of text in the field and using the table UNPK:TBL to convert it to a four-byte character word. The character word may carry a live text character or punctuation and/or control information (data link escape, subfield or filed delimiters) and diacritical or underscore flag bits.

After the above fields are unpacked and linked to their associated call number, and heading information is stored in the heading pointer table, the processing indices for fields which may be traced are interrogated. Each tracing is unpacked in trun for the fields whose indices show they should be traced. Before processing of the catalog card begins, the sort tag segments for the main entry (or title as main entry) and the title (or uniform title) are created using the routine FMT:TAG. Then a read is issued for the next input record while card processing continues on the current record.

At RESTART the totals for cards and records for a request are cleared as well as previous indention values. Then at NEXTHD the next heading address is interrogated in the heading pointer table, P:HEADS. If all headings have been processed for the current input transaction record, output totals are incremented; and control goes back to FMT:INIT to process the next transaction record. Otherwise the pointer to the PDT receiving catalog option tables (SPCRs) is reset to the first SPCR and processing begins for the next set of cards. At NEXTSPCR the next SPCR is parsed and the options for the current receiving catalog are set up such as type of card (main entry, shelf list, other), type of heading (top-of-card, marked, dropped), extra card and indention values. The SPCR options are checked against the type of card requested in the input record to determine whether a card should be produced for this receiving catalog. If not, control goes back to NEXTSPCR. Otherwise the input fields are

ready to be formatted into paragraphs on the catalog card. When all the SPCR's have been parsed, control returns to NEXTHD to look at the next heading required.

The first step in formatting the catalog card is building the remainder of the bottom line, or authority line, by adding the NUC symbol and the accession number, if present, and the receiving catalog, or pack, I.D.. Then the type of heading specified in this SPCR is built. If the heading type is zero, the heading desired is a top-of-card heading. If the heading type is not zero, a branch is taken to HD:TYPE1. If the Type0 heading is associated with a main entry or shelf list card request, no heading is needed and a branch is taken to HD:910. Otherwise the heading is built in the following manner.

If the heading request is a normal one, the heading indicated by the pointer in the table P:HEADS is unpacked as the top-of-card heading. If the request is for a series statement heading, the heading indicators are checked. Otherwise a branch is taken to HD:50. If the author is the main entry, the main entry is used as the heading, and the author may be separated from the title. If the author is not the main entry, and this is a title statement heading request, the special exit EXIT:2 is set up at HE:50 for the routine FMT:CC to unpack only the first subfield as the heading. If this is not a title statement heading request, special exit EXIT:3 is set up for the routine FMT:CC to separate the author from the title.

After the proper top-of-card heading is unpacked, unnecessary trailing punctuation is deleted. Then the sort tag segment for the heading is built. The heading is capitalized if desired and the heading is seppacked for output. If the heading is too big (vertically) the minor error switch, MINOR:SW, is set and restart is begun for the entire input transaction record at RESTART. If there has already been a minor error for this transaction, a message is printed, the heading is deleted, and processing continues on the remainder of this card.

At HD:TYPE1 a Type1 marked dropped heading is built. Then the dropped heading designator from the PDT is inserted before the packed heading. The heading is capitalized, if desired and control goes to HD:910. If the heading is Type2 (dropped, capitalized) heading, the heading is capitalized but not marked.

At HD:910 the card text line and current line numbers are set. Then the sort tag segments are formatted into a complete sort tag using the routine FMT:ORDER and inserted in the output record along with the sequential number of this record within the output fill and the billing code for the requesting institution. The bottom line is seppacked into the output record following the record leader. Then the top-of-card heading is moved into the output buffer. Each text paragraph is now seppacked in the temp area in preparation for its insertion into the output record. The seppack process involves interpreting each character word in the text of a field and placing the five text bytes in the card paragraph along with diacriticals and underscore control information. Tab

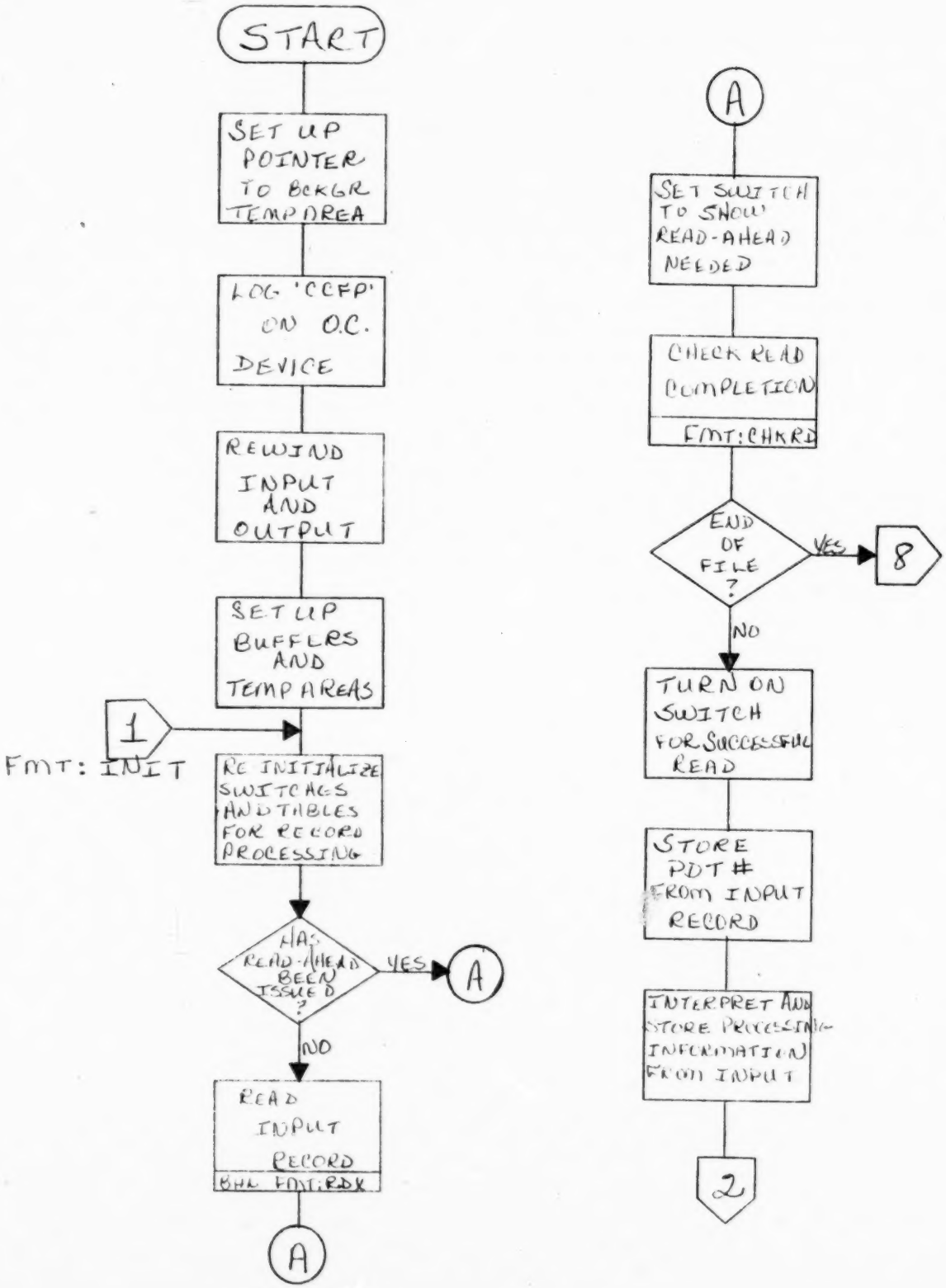
values for indentions and the rod hole line are inserted and the paragraph is broken down into card lines. The paragraph beginning line number on the entire card including continuation is stored as the first byte of each paragraph. The call numbers are positioned on the card in the proper relation to their associated paragraphs. If a continuation card is needed, a continuation policy is built and seppacked. The routines FMT:SPK, FMT:SP1, FMT:SP2, FMT:SPCN, and FMT:SPK are used in the seppack process. After all fields except the tracings are seppacked, the tracings or Type2 or 3 dropped heading (if present) is seppacked.

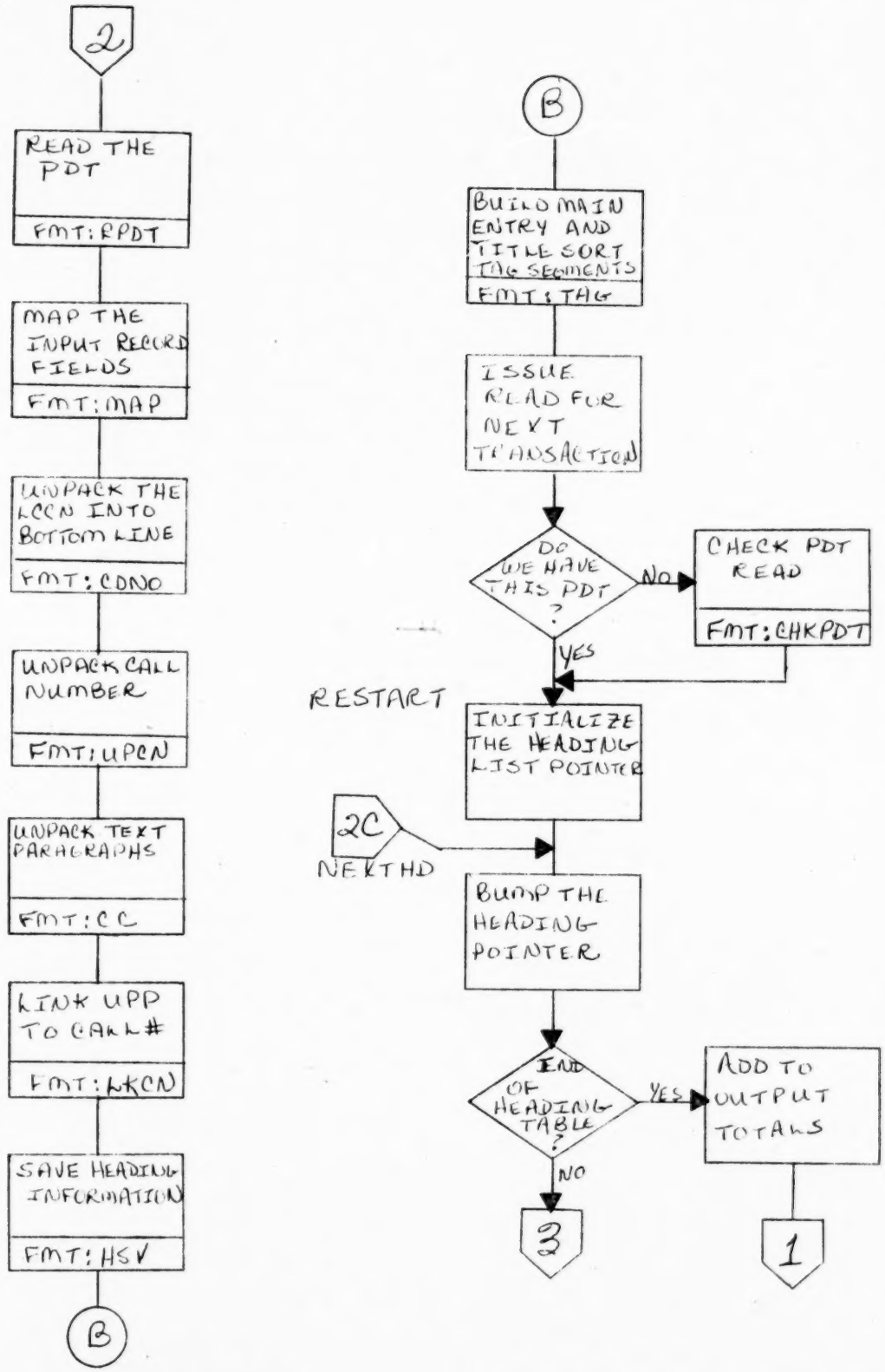
At AE:SP10 an attempt is made to push the tracings paragraph toward the bottom of the last card of the set. If the tracings are as far down as they can go, a branch is taken to END:SP. If the tracings paragraph can be pushed down to below the rod hole without spanning cards and without reseppacking the paragraph beginning line number is adjusted and control goes to END:SP. Otherwise the unpacked tracings paragraph is rescanned to see if it can be vertically adjusted on one card. If so, the switch VFC:AE1 is set. Then the paragraph is reseppacked, and if VFC:AE1 is still set (paragraph was reseppacked on one card) control returns to AE:SP10 to try pushing the tracings down some more. Otherwise control falls through to END:SP. At END:SP the pointer to the first text paragraph within the output record is stored in the record leader. Then the seppacked paragraphs are inserted in the output record in the following order: main entry, uniform title (or title), series statement, special notes (399 or 499), biblio notes, special notes (599), tracings (or dropped heading), and continuation policy. The routine used to insert the paragraphs in the output record FMT:INSP inserts a call number associated with a paragraph before the paragraph in the output record.

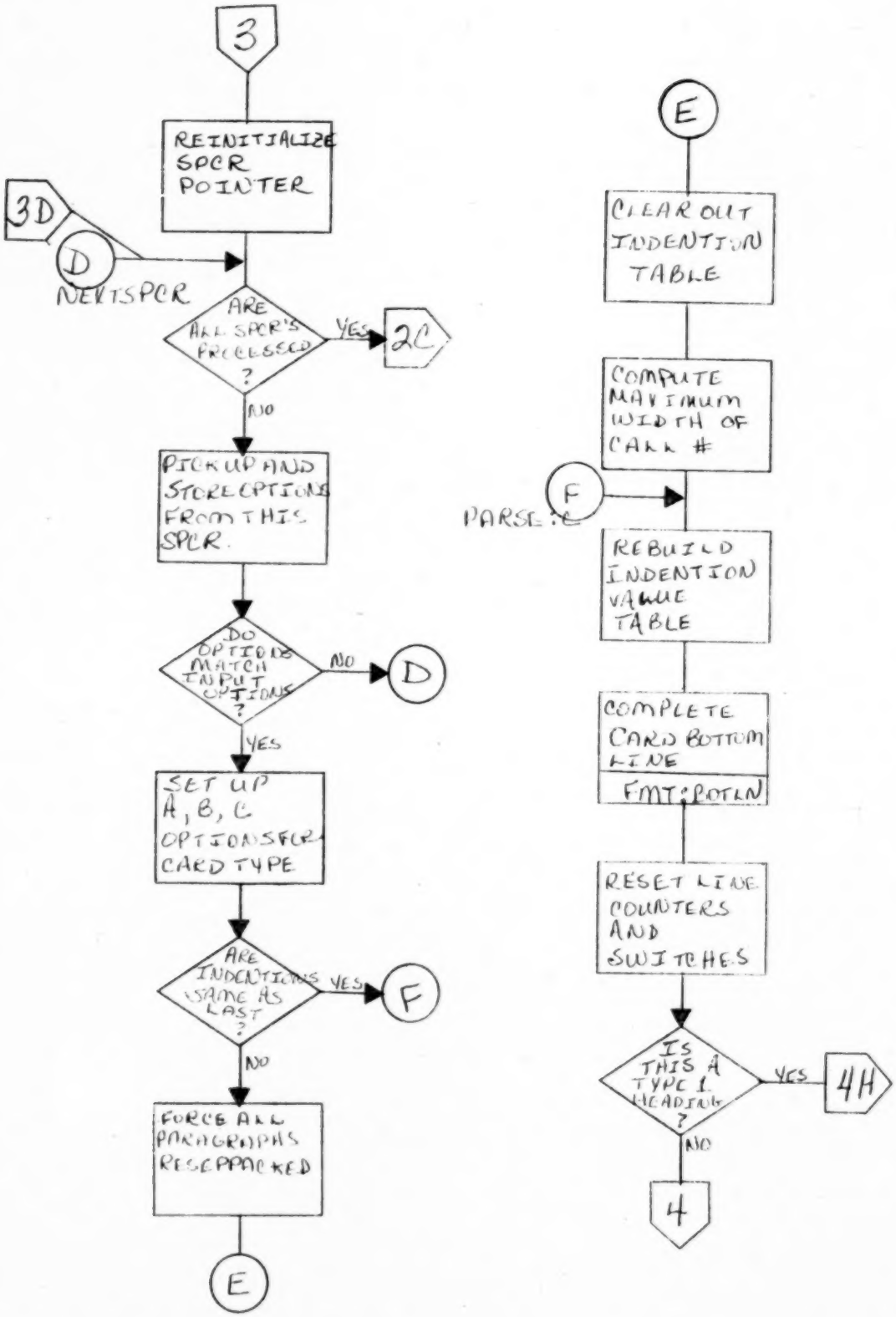
After the record has been built the total number of cards this record will produce is calculated and stored in the output record along with the continuation line number if a continuation is present. Then the record length is set up, and the record is written to the output device. After the record is output if the heading was Type1 (dropped, marked), the heading mark is wiped out. The heading is decapitalized if it was capitalized, and the tracings paragraph is marked to be reseppacked. If the heading was Type2, it is decapped and marked for reseppacking. Then control returns to NEXTSPCR to see if more cards are to be produced for this set.

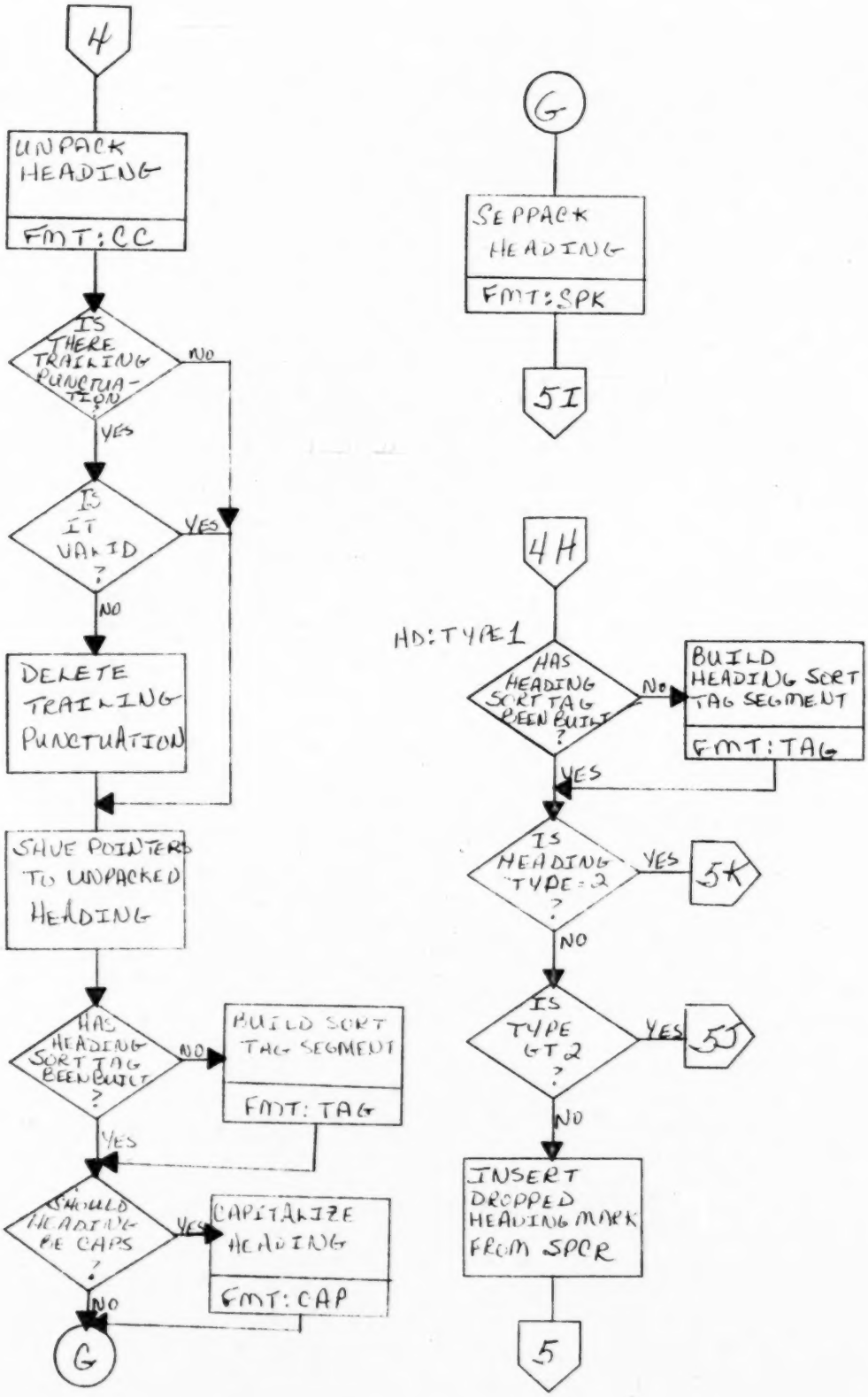
After all the input transaction records have been processed, control goes to FMT:FIN. Here a 22-word control record is written as the last record of the output file. It contains in the 22nd word the total number of cards produced during this run. Then the input and output volumes are unloaded and CCFP statistics are calculated and printed.

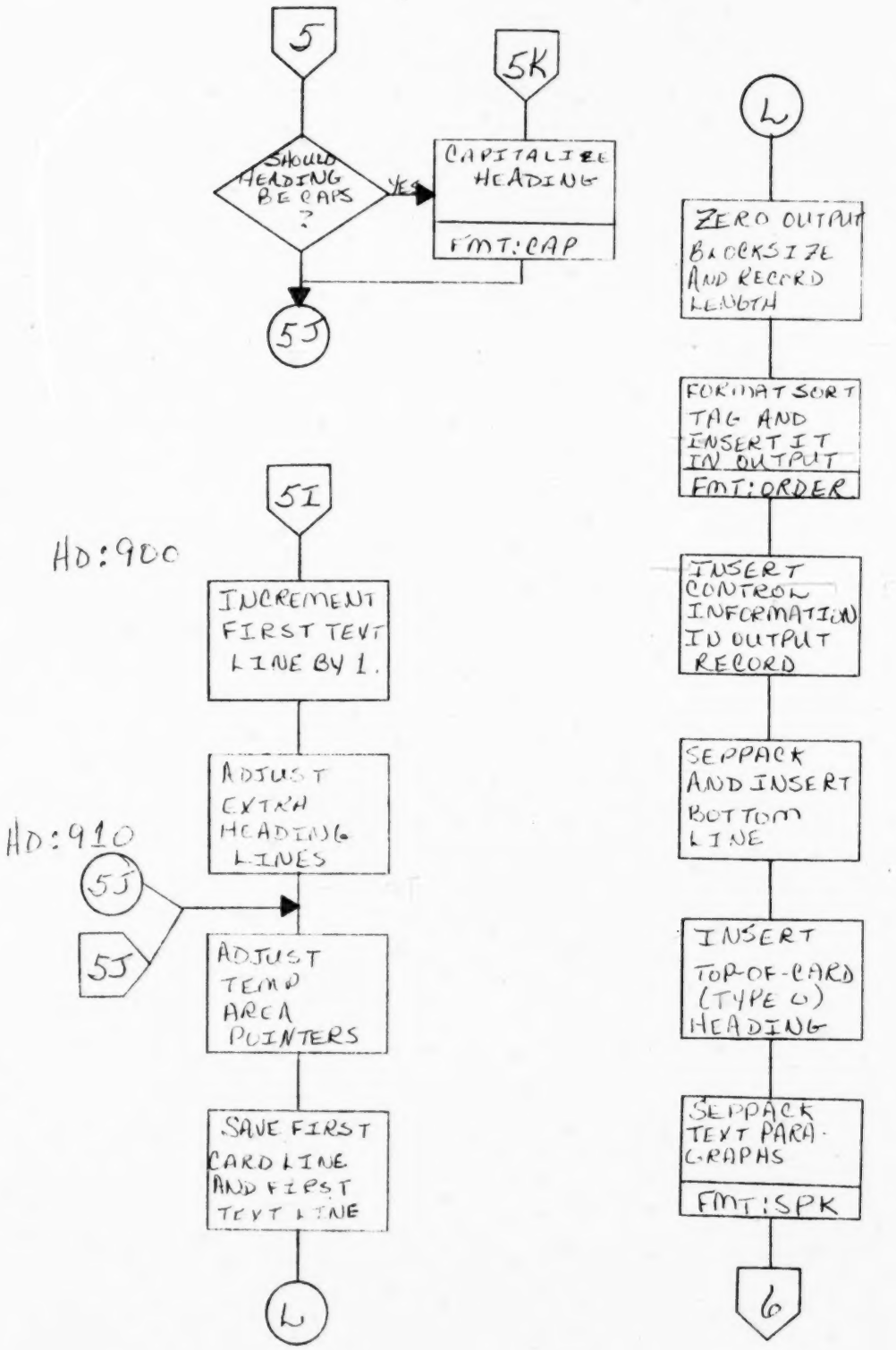
V. GENERAL INFORMATION FLOW

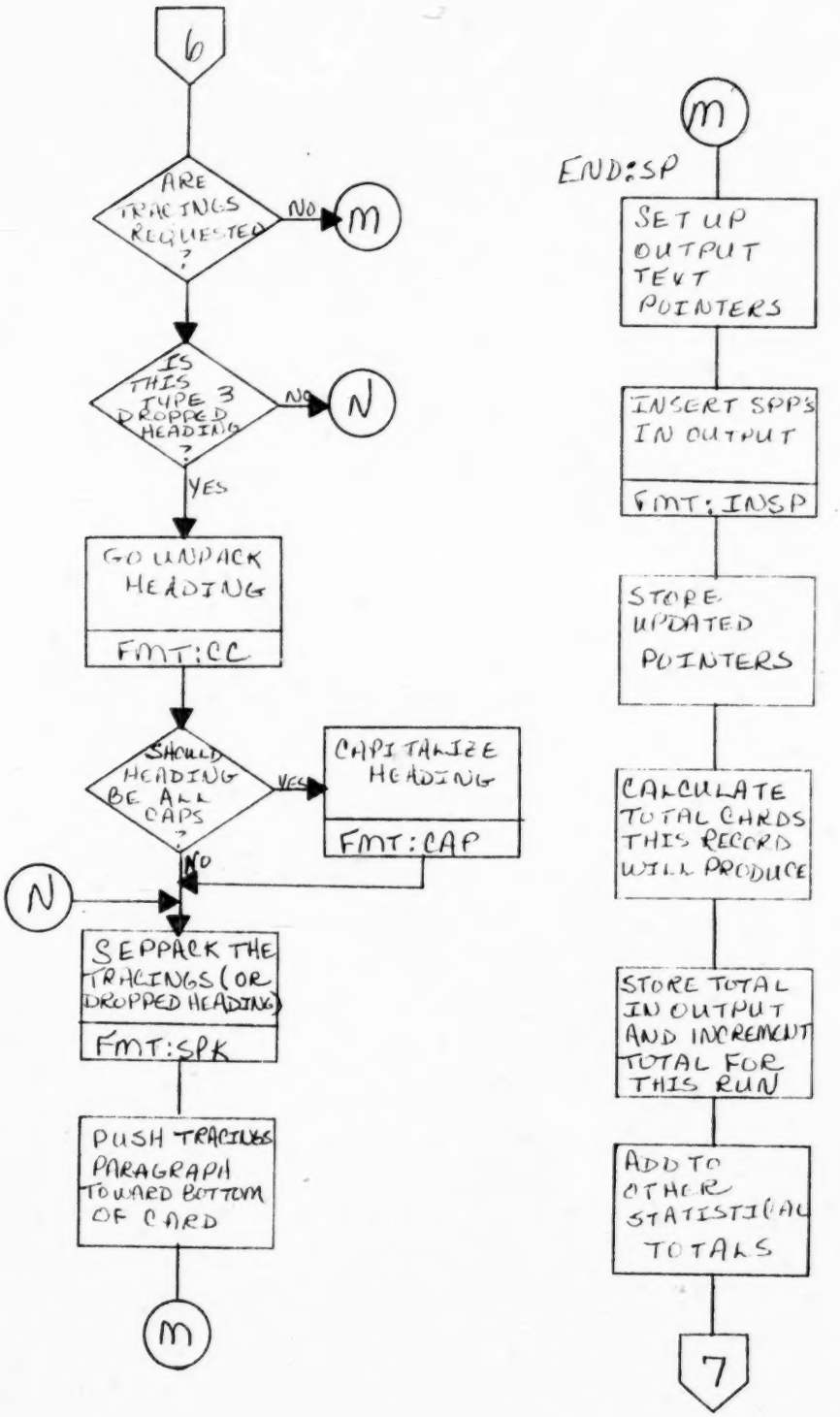


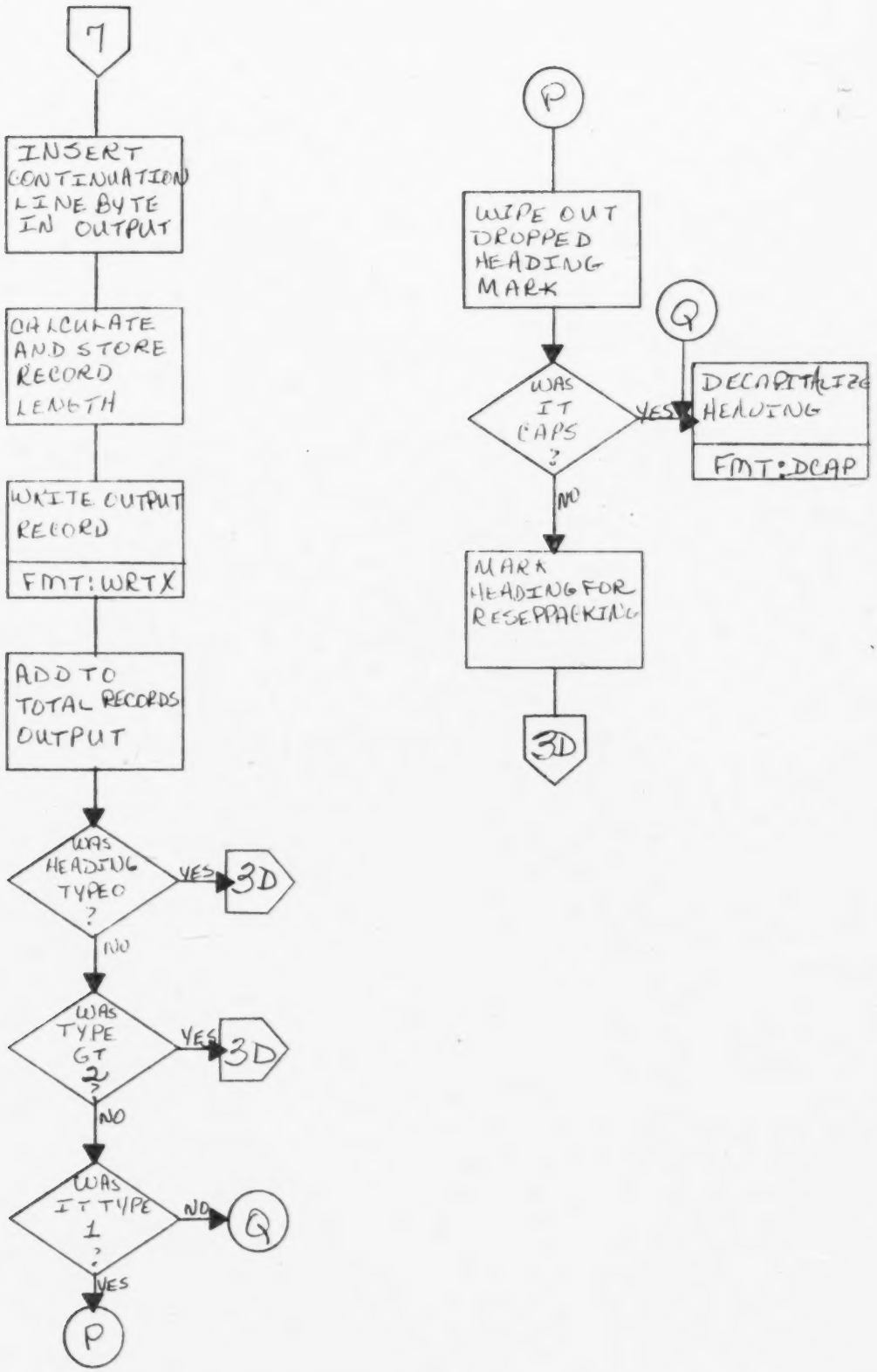

2
READ THE PDT
FMT:RPDT
MAP THE INPUT RECORD FIELDS
FMT:MAP
UNPACK THE LCCN INTO BOTTOM LINE
FMT:CONO
UNPACK CALL NUMBER
FMT:UPCN
UNPACK TEXT PARAGRAPHS
FMT:CC
LINK UPP TO CALL#
FMT:LKCN
SAVE HEADING INFORMATION
FMT:HSV
B
B
BUILD MAIN ENTRY AND TITLE SORT TAG SEGMENTS
FMT:TAG
ISSUE READ FOR NEXT TRANSACTION
DO WE HAVE THIS PDT ?
NO
CHECK PDT READ
FMT:CHKPDT
YES
RESTART
INITIALIZE THE HEADING LIST POINTER
2C
NEXTHD
BUMP THE HEADING POINTER
END OF HEADING TABLE ?
YES
ADD TO OUTPUT TOTALS
NO
3
1

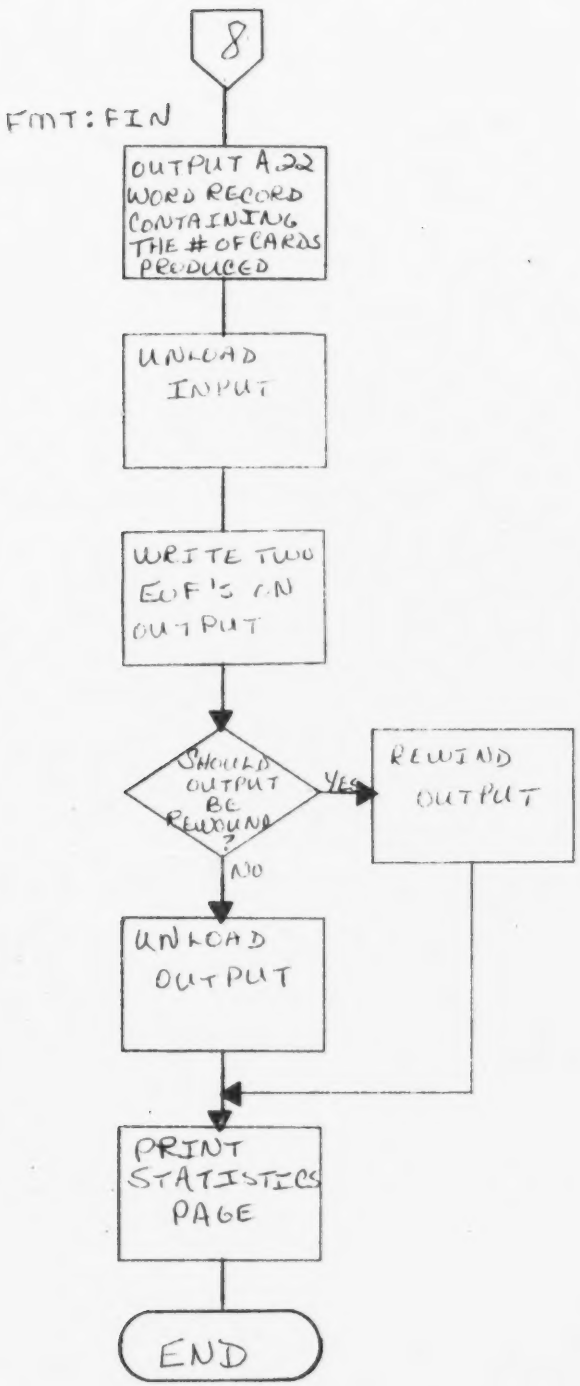

3
REINITIALIZE SPCR POINTER
3D
D
NEXTSPCR
ARE ALL SPCR'S PROCESSED ?
YES
2C
NO
PICK UP AND STORE OPTIONS FROM THIS SPCR.
DO OPTIONS MATCH INPUT OPTIONS ?
NO
D
YES
SET UP A, B, C OPTIONS FOR CARD TYPE
ARE INDENTIONS SAME AS LAST ?
YES
F
NO
FORCE ALL PARAGRAPHS RESEPARATED
E
E
CLEAR OUT INDENTION TABLE
COMPUTE MAXIMUM WIDTH OF CARD #
F
PARSE:E
REBUILD INDENTION VALUE TABLE
COMPLETE CARD BOTTOM LINE
FMT:BOTLN
RESET LINE COUNTERS AND SWITCHES
IS THIS A TYPE 1 HEADING ?
YES
4H
NO
4


4
UNPACK HEADING
FMT:CC
IS THERE TRAILING PUNCTUATION?
NO
YES
IS IT VALID?
YES
NO
DELETE TRAILING PUNCTUATION
SAVE POINTERS TO UNPACKED HEADING
HAS HEADING SORT TAG BEEN BUILT?
NO
BUILD SORT TAG SEGMENT
FMT: TAG
YES
SHOULD HEADING BE CAPS?
YES
CAPITALIZE HEADING
FMT: CAP
NO
G
G
SEPPACK HEADING
FMT:SPK
5I
4H
HD:TYPE1
HAS HEADING SORT TAG BEEN BUILT?
NO
BUILD HEADING SORT TAG SEGMENT
FMT:TAG
YES
IS HEADING TYPE=2?
YES
5K
NO
IS TYPE GT 2?
YES
5J
NO
INSERT DROPPED HEADING MARK FROM SPCR
5


5
SHOULD HEADING BE CAPS ?
YES
5K
CAPITALIZE HEADING
FMT:CAP
5J
L
ZERO OUTPUT BLOCKSIZE AND RECORD LENGTH
FORMAT SORT TAG AND INSERT IT IN OUTPUT
FMT:ORDER
INSERT CONTROL INFORMATION IN OUTPUT RECORD
SEPPACK AND INSERT BOTTOM LINE
INSERT TOP-OF-CARD (TYPE 0) HEADING
SEPPACK TEXT PARAGRAPHS
FMT:SPK
6
5I
HD:900
INCREMENT FIRST TEXT LINE BY 1.
ADJUST EXTRA HEADING LINES
HD:910
5J
5J
ADJUST TEMP AREA POINTERS
SAVE FIRST CARD LINE AND FIRST TEXT LINE
L


6
ARE TRACINGS REQUESTED ?
NO
M
IS THIS TYPE 3 DROPPED HEADING ?
NO
N
YES
GO UNPACK HEADING
FMT:CC
SHOULD HEADING BE ALL CAPS ?
YES
CAPITALIZE HEADING
FMT:CAP
NO
N
SEPPACK THE TRACINGS (OR DROPPED HEADING)
FMT:SPK
PUSH TRACINGS PARAGRAPH TOWARD BOTTOM OF CARD
M
M
END:SP
SET UP OUTPUT TEXT POINTERS
INSERT SPPS IN OUTPUT
FMT:INSP
STORE UPDATED POINTERS
CALCULATE TOTAL CARDS THIS RECORD WILL PRODUCE
STORE TOTAL IN OUTPUT AND INCREMENT TOTAL FOR THIS RUN
ADD TO OTHER STATISTICAL TOTALS
7


7
INSERT CONTINUATION LINE BYTE IN OUTPUT
CALCULATE AND STORE RECORD LENGTH
WRITE OUTPUT RECORD
FMT:WRTX
ADD TO TOTAL RECORDS OUTPUT
WAS HEADING TYPE 0 ?
YES
3D
NO
WAS TYPE GT 2 ?
YES
3D
NO
WAS IT TYPE 1 ?
NO
Q
YES
P
P
WIPE OUT DROPPED HEADING MARK
Q
WAS IT CAPS ?
YES
DECAPITALIZE HEADING
FMT:DCAP
NO
MARK HEADING FOR RESEPPACKING
3D


8
FMT:FIN
OUTPUT A 22 WORD RECORD CONTAINING THE # OF CARDS PRODUCED
UNLOAD INPUT
WRITE TWO EOF'S ON OUTPUT
SHOULD OUTPUT BE REWOUND ?
YES
REWIND OUTPUT
NO
UNLOAD OUTPUT
PRINT STATISTICS PAGE
END

VI. SOFTWARE INTERFACE

A. Linkage - Background Linkage with OBM

B. Parameter List Description - none

C. Return Codes- none

D. Other Entry Points - none

E. OCLC External Subroutines Referenced - none

F. OCLC Procedures Referenced

PUSH	GENNUL
PULL	GENNONE
RESTORE	GENN
SAVE	GEND
@RDTY	GENBLK
@WTTY	GENA1
@PRINT	GENA0
@BINBCD	GEN:LG2
@BCDHEX	GEN:LG1
@BCDBIN	GEN:
@ERROR	CALL
TTBL	BRB
SAVE	BPUN
ROMAN	BNUM
RETURN	BNRB
PTBL	BNPUN
OTBL	BNNUM
OPTB	BNLB
ITBL	BNB
GENVH	BNALP
GENUND	BLB
GENSPEC	BB
GENP	BALP

VII. SPECIAL STORAGE AREAS, SWITCHES, AND TABLES

A. Special Storage Areas

PDT BUFFER- The Profile Definition Table (PDT) read from disk (see III. SUMMARY OF INPUT AND OUTPUT) is expanded in the PDT buffer, PDT:L. The result is the following leader and SPCR option tables:

bytes	0 - 3	PDT number
	4 - 7	number of SPCRs that follow
	8 - 11	institution code
	12- 15	cataloging library code
	16- 19	holding library code
	20- 31	X'00's (unused)
	32- 131	SPCR #1
SPCR:	0 - 4	pack I. D.
	5 - 7	billing code
	8 - 9	pointer to NUC symbol at end of PDT
	10- 11	pointer to PDT stamp at end of PDT
	12- 15	indentions 1,2,3,4
	16- 16	sort order code
	17- 17	dropped heading mark
	18- 18	maximum lines a top-of-card heading may occupy
	19- 19	X'00' (unused)
	20- 85	22 3-byte card options (padded with zeros)

Byte-0 A options
- A_1-type of non-PDT stamp allowed
  type of holdings allowed
- A_2-type of tracings allowed
  type of user data allowed

Byte-1 B options
- B_1-biblio contents notes indicator
  biblio notes (500's) indicator
- B_2-heading type indicator
  heading capitalization

Byte-2 C options
- C_1 - C_4-indentions 1-4

	86- 99	X'00's (unused)
bytes	100n+31	SPCR #n
	100n+131	NUC symbol length (NL)
	100n+132	NUC symbol
	100n+132+NL	PDT stamp length
	100n+133+NL	PDT stamp

SEPPACKED PARAGRAPH- A seppacked paragraph consists of the text for a paragraph including diacritical marks and broken down into card lines. Tab values are present as well as the absolute line number of the line on which the paragraph begins. Each seppacked paragraph is preceded in the temp area by a leader which carries the paragraph's status at the last entry to and exit from the seppack subroutine, FMT:SPK.

SPP leader:

- HW-0 exit CUR:LN (the current line number being used on the card)
- HW-1 entry CUR:LN
- HW-2 exit PREV:CLS (total lines used on previous cards in this set)
- HW-3 entry PREV:CLS
- HW-4 last line number on card used by the call number
- HW-5 len(seppacked call number)
- HW-6 current CUR:FLTX (first text line number for extension cards)
- HW-7 byte length of seppacked paragraph (not including leader)
- FW-4 WA(seppacked call number)
- HW-10 status of paragraph (meaning if BITn=1)
  - BIT-0 paragraph spans two or more cards
  - BIT-1 paragraph intersects the rod hole
  - BIT-2 paragraph ends with a short line (one which does not reach the rod hole)
- HW-11 beginning of temp area which next paragraph may use

SPP: The first byte of each seppacked paragraph (SPP) is the absolute line number of the line on the card where the paragraph begins. (The line number is the second byte of the call number paragraph). Line number one is the first line of the first card. The remaining bytes are either text or control bytes.

SPP control bytes:

- X'FF' end of SPP or beginning of call number SPP
- X'FE' end of card line within paragraph
- X'FD' tab value follows
- X'FC' diacritical follows

B. Switches

GOTRECSW-initialized to -1, this switch is set to +1 to indicate that an input record has been read.

GOTPDTSW-initialized to -1, this switch is set to +1 to indicate that a PDT has been read or is still in PDT:L from the last transaction.

SW:HSV-the heading save information switch is initialized to zero. It is set to non-zero on return from the heading information save routine, FMT:HSV, whenever a heading is to be kept and printed.

READSW-this switch is used to indicate whether or not a read-ahead is to be issued for the next input transaction record. READSW=-1, read-ahead needed. READSW=+1, read-ahead has been issued.

MINOR:SW-indicates that one or more minor errors have occurred during formatting.

TGBUILT-set to -1 initially. If zero, the sort tag segment for the heading has been built.

C. Tables

MAP:TBL, MAP:ITBL, MAP:PTBL- These tables are used by the subroutine, FMT:MAP, to identify the tags in the input record and specify some processing indices and the correct map pointer table to use. MAP:TBL is a table of tag numbers which is searched for an input tag using the binary method. When a matching tag is found, its displacement in MAP:TBL is used to pick up an index from MAP:ITBL which is then used in referencing the table, MAP:PTBL. Each MAP:PTBL entry consists of three elements:

AF(1) unique statement identifier
AF(2) index into an SPCR which specifies how the card associated with a particular type of entry is to be formatted.
AF(3) address of the map pointer table where the input field address will be saved.

CC:OPTBL, CC:PBF1, CC:PBF2, CC:PSF, CC:PEF- These tables are used by the field concatenation subroutine, FMT:CC, to direct the formatting of the unpacked paragraphs (UPPs). CC:OPTBL is a list of processing options for the various input fields. CC:OPTBL is indexed using the processing indices from MAP:PTBL. Each entry in CC:OPTBL consists of four bytes: a one-byte index into each of the tables CC:PBF1, CC:PBF2, CC:PSF, CC:PEF. The tables consist of routines within FMT:CC that will process respectively: beginning of field, additional beginning of field, between subfield, end of field.

MAP POINTER TABLES- Each map pointer table has a label beginning "P:". It contains in WORD0 the address of the field in the input record. WORD1 contains the processing

indices from MAP:PTBL.

UPP POINTER TABLES- Each unpacked paragraph (UPP) pointer table has a label beginning "UP:". It consists of two to five words which contain the following information:
WORD-0 temp area address of UPP
WORD-1 temp area address of end of UPP
WORD-2 end of short title address
WORD-3 address of UPP's SPP
WORD-4 UP call number's address

P:HEADS- A table of pointers to headings in the input transaction record for which cards are to be produced. The entry for each heading consists of four words:
WORD-0 address of packed heading in input record
WORD-1 processing indices from MAP:PTBL
WORD-2 beginning of unpacked heading in temp area
WORD-3 end of unpacked heading in temp area

T:LANG- A table of pointers to the resident foreign language tables. T:LANG is indexed using the language code in the input record leader.

A:LIST- A table of English articles

ABLIST, ADLIST- ABLIST is a table of common abbreviations. When an abbreviation is found in ABLIST, the index to ABLIST is used to index ADLIST to find the complete word or phrase.

OPTB:TBL- A table of halfword entries. Each entry consists of four four-bit indicators used in processing the heading. The table is indexed by the B_2 option from the SPCR options for the current card.
AF(1) heading type
0- top-of-card
1- dropped inplace (marked)
2- dropped inplace (capped)
3- dropped stand-alone
AF(2) capitalization
0- upper and lower case
1- all upper case
AF(3) heading structure
0- normal
1- main entry, shelf list, unit cards
2- title cards
3- author and title separated
AF(4) heading concatenation method

UNPK:TBL- A table of character words, punctuation words, and control words to be used by the unpack subroutine, FMT:UNPK. The table is indexed by the EBCDIC value of the live text byte.

UNPK:TG- A table of Greek character words.

UNPK:TP, UNPK:TB - Tables of superscript and subscript character words respectively.

PUN:LIST- A table of acceptable ending punctuation for headings and continuation policy.

LG: TABLES - Several tables of foreign language articles.

P:SPCR - A table of offset pointers to the different elements of the SPCR.

U: TABLE - A table of special unpacked characters, (dash, hyphen, Roman numerals, etc.)

INDI:VAL - A table of indention values.

VIII. APPENDIX

APPENDIX A

OPERATING REQUIREMENTS

1. Computer - Xerox Sigma 5

2. I/O Devices - 2 9-track tape drives, line printer, card reader.

3. Operating System - OBM

4. Execution Time -

5. Run Schedule - daily

6. Job Control Language

```
!JOB      ONLINE, FORMAT
!IMG002A  F1
!PAU - MOUNT 2ND SCRATCH TAPE ON T0, RING IN
!POOL 0
!ASSIGN (M:BI,T1), BIN, (TRIES, 10)
!ASSIGN (M:BO,T0), BIN, (TRIES, 10)
!PMD
!CCFP6
!STD  (C, D6, SORT)
!JOB IDLE,
!IMG002A  F2
!IMG002A  CCR
!FIN
```

APPENDIX B

OPERATING CHARACTERISTICS

1. CONSOLE MESSAGES

MESSAGES:

!!PAU -- MOUNT 2nd SCRATCH TAPE ON T0, RING IN *

RESPONSE: Mount tape as directed
'C'
NEW LINE

MESSAGE:

CCFP
RESPONSE: None

2. PROGRAMMED ABNORMAL COMPLETION -

CCFP will terminate abnormally via a CALl, 9 3 instruction under the following conditions. The message printed out to signal the abort is included in each case.

MESSAGE	REASON
E10 - EEAD ERROR ON BI DEVICE	A read error occurred when reading the card input record in FMT:RDX.
E19 - WRITE ERROR ON BO DEVICE	A write error occurred when writing the formatted card image onto BO device in the routine, FMT:WRTX.
E21 - BI BUFFER OVERFLOW	The card input record overflows the buffer in the routine, FMT:RDX.

3. DIAGNOSTICS - ERRORS

CCFP will reject the record and take a direct branch to FMT:INIT under the following conditions.

MESSAGE	REASON
E1 - FIELD 000 is ILLEGAL	The input tag is not found in field tag table.
E2 - FIELD 000 OVER-FLOWED GROUP POINTER TABLE	End of field pointer table is encountered before an empty slot is found.
E3 - TARGET AREA TOO SMALL	The target Area for the unpacked paragraph is too small.
E4 - SNYTAX ERROR IN INPUT RECORD	A 'DELTA', a pound sign, or any other illegal character is found within a field.

MESSAGE	REASON
E5 - INVALID EXIT FROM CC ROUTINE	The routine, FMT:CC determined that the record should be deleted.
E6 - MISSING TITLE PARAGRAPH	There is no 245 field in the record.
E7 - TEMP CORE RUNOUT	The seppack routine has run out of core space.
E8 - OUTPUT BUFFER OVERFLOW	The routine FMT:INSP has run out of room in the output buffer.
E9 - WORD TOO BIG TO FIT A SINGLE LINE	One word cannot be longer than the width of the card.
E11 - HEADING POINTER TABLE OVERFLOW	The heading pointer table is full upon entry.
E12 - DROP HEADING SOFTWARE ERROR	CCFP was unable to pick up the pointer for the dropped heading from the table 'P:HEADS'.
E13 - BOTTOM LINE ERROR	There is a snytax error in the packed text for the bottom line.
E14 - PERM READ ERROR ON PDT DEVICE	The routine, FMT:RPDT was unable to read the requested PDT.
E15 - MISSING OR INCORRECT PDT	The routine, FMT:RPDT was unable to find the PDT or the PDT was not correct.
E24 - CN SYNTAX ERROR	2 possible reasons: 1. subfield code was missing from table. 2. syntax error after unpacking the call number.
E25 - TEMP AREA RUNOUT	Temporary buffer overflowed when unpacking the call number.
E26 - CN TEMP AREA RUNOUT	The temporary area for seppacking the call number has overflowed.
E27 - SPCN ERROR	An error as occurred in seppacking the call number.
E28 - UPCN ERROR	After unpacking, the call number has an illegal subfield code.
E29 - CONPOL TOO BIG FOR A SINGLE CARD	The continuation policy will not fit on one card after being re-seppacked.
E30 - NO ROOM FOR SP. NOTES IN CN AREA	A usuable continuation policy for the call number has not been formulated.
E32 - ILLEGAL EXTENDED SET DESIGNATOR	The character set identifier could not be found in table.

MESSAGE	REASON
E33 - FIELD 000 IS INCORRECTLY TERMINATED	An invalid field delimiter is found in the record.

3. DIAGNOSTICS - WARNING

MESSAGE	ACTION
W16 - HEADING TOO LARGE	The heading is deleted.
W17 - CALL NO. TOO BIG	Entire paragraph is re-seppacked.
W18 - ***	NOT USED NOW
W22 - PDT READ ERROR	NOT USED NOW
W31 - UNPRINTABLE SYMBOL - X' '	The Next source byte is unpacked.

4. PARAMETER CARDS REQUIRED - None

5. EXAMPLE OF OUTPUT - NEXT PAGE

OCLC CATALOG CARD FORMATTING PROGRAM 19:33 DEC 05,'72 PAGE 1

RUN STATISTICS

LENGTH OF BG TEMP AREA IS 25194

NO. INPUT XACTIONS: 1061

NO. OUTPUT RECORDS: 7578

AVE. BYTES PER RECORD: 507

MAX. RECORD SIZE: 1970

TOTAL CARDS: 7879

MAX. EXTENTION CARD NO.: 3

EXTENT. CARD FREQ. DIST.:

0: 7299 1: 263 2: 18 3: 0 4: 0 5: 0 6: 0 7: 0 8: 0 9+: 0

AVE. CARDS PER XACTION: 7.42

SECS. WALL CLOCK TIME: 198

SECS. PROCESS TIME: 153.20

AVE. MSECS. PER CARD: 19.5

NO. BF ERRORS: 0

VIII.6

APPENDIX C

DETAILED DESCRIPTION OF INTERNAL SUBROUTINES

ROUTINE	PAGE
AE:1	VIII.8
BCD:BIN	VIII.78
BCD:HEX	VIII.77
BIN:BCD	VIII.75
BIN:BCDR	VIII.76
FMT:BOTLN	VIII.53
FMT:CAP	VIII.28
FMT:CC	VIII.19
FMT:CDNO	VIII.52
FMT:CHKRD	VIII.67
FMT:DCAP	VIII.29
FMT:ERR	VIII.62
FMT:HSV	VIII.24
FMT:INSN	VIII.47
FMT:INSP	VIII.56
FMT:LKCN	VIII.14
FMT:MAP	VIII.10
FMT:ORDER	VIII.54
FMT:PRT	VIII.57
FMT:RDTY	VIII.63
FMT:RDX	VIII.65
FMT:RPDT	VIII.68
FMT:SP	VIII.30
FMT:SPCN	VIII.43
FMT:SPK	VIII.33
FMT:SPSP	VIII.46
FMT:SPTS	VIII.48
FMT:TAG	VIII.49
FMT:UNL	VIII.74
FMT:UNPK	VIII.15
FMT:UPCN	VIII.12
FMT:UPCP	VIII.25
FMT:WRTX	VIII.71
FMT:WTTY	VIII.64
FMT:WXREW	VIII.74
FMT:WXWEOF	VIII.73
SP:1	VIII.9
SPK:CP	VIII.40
SPK:ERCN	VIII.42

I. FUNCTIONS

The purpose of AE:1 is to act as a driver for builing unpacked tracings paragraph entries. Upon entry to the routine, the processing indices for the entry are picked up from the input map pointer table. The routine FMT:HSV is used to store heading information in the heading pointer table. The routine FMT:CC is then used to unpack the entry. Upon return from FMT:CC, two blanks are stored following the unpacked entry. Then the next input map pointer table is checked. If the first word is less than zero, control is returned to the calling routine. Otherwise the next entry is processed.

II. SOFTWARE INTERFACE

A. Linkage

```
BAL,R7     AE:1
```

B. Parameter List Description

Upon entry to AE:1 the general registers must be set as follows:

R6 - points to (the input map pointer table for this entry) +1.

R8 - contains the character word for an unpacked blank.

C. Return Codes - none

D. Other Entry Points - none

E. OCLC CCFP Routines Referenced

FMT:HSV
FMT:CC

F. OCLC Procedures Referenced - none

I. FUNCTIONS

The purpose of SP:1 is to act as a driver to unpack the special notes paragraphs. Upon entry the field indicator byte is picked up from the input map pointer table and stored in Word 2 of the unpacked paragraph (UPP) pointer table. The beginning address of the unpacked paragraph also is stored in the pointer table in Word 0. Then the processing indices a picked up from the input map pointer table, and the routine FMT:CC is used to unpack the field. Upon return from FMT:CC the ending address of the unpacked paragraph is stored in the UPP pointer table. Then the next packed field entry in the map pointer table is checked. If the first word is -1, control is returned to the calling routine. Otherwise the next special notes field is unpacked.

II. SOFTWARE INTERFACE

A. Linkage

BAL,R14 SP:1

B. Parameter List Description

Upon entry to SP:1,R6 and R7 should be initialized as follows:

R6 - contains the address of the packed field entries in the map pointer table.

R7 - contains the address of the UPP pointer table entries.

C. Return Codes - none

D. Other Entry Points - none

E. OCLC CCFP Routines Referenced

FMT:CC

F. OCLC Procedures Referenced - none

I. FUNCTIONS

This routine maps all fields of the input transaction record into a set of pointer tables associated with the respective field types. Also, field processing indices, which later determine the field processing method as a field within a paragraph and as a heading - are inserted in these tables adjacent to the field addresses.

Upon entry to FMT:MAP, the tag of the first variable length field is set up in R2, and the length of the field is set up in R3. A binary search is then performed on the field tag table, MAPTBL. If the input tag is not found in the table, an error is declared and the transaction record is rejected. When the tag is found, the index into the table of pointer table addresses (MAP:PTBL) is located in MAP:ITBL. Then the address of the field pointer table to be used is loaded from MAP:PTBL into R5, and the processing indices for the field are loaded into R4.

The field pointer table is searched for an empty slot. If the end of the pointer table is encountered (denoted by a word = -3), an error is declared and processing for the current record is terminated. When a -1 is encountered (denoting the next available slot in the table), the link switch, MAP:LKSW, is checked. If the link switch is set, the link from the previous field is "or'ed" into the processing indices in R4 and MAP:LKSW in reset. Then the byte address of the field is stored in the first word of the slot, and the field processing indices in R4 are stored in the second word of the pointer table slot. A-1 is stored after the indices denoting the next available slot. If the field just stored was a call number, a backward call number link is set up for the next field and the link switch is turned on.

After the pointer table entry has been built, the processing indicators in R4 are checked for a main entry field. If this is a main entry field (tag value 1xx) the field indicators are checked. If the second indicator is one, the 1xx tag is changed to 6xx, and the new tag and old indicators are stored in the pointer table for the 6xx field. If the field is not a main entry, the end of the field is checked for a valid delimiter. If the end of the record is found, the map is complete and control is returned through R14. If an end-of-field delimiter is found, the next field is mapped. If an invalid delimiter is found, an error is declared and processing for the current record is terminated.

II. SOFTWARE INTERFACE

A. Linkage

Call is

```
BAL,R14    FMT:MAP
```

B. Parameter List Description

Before branching to FMT:MAP the byte address of the input record must be stored at P:XREC.

C. Return Codes - none

D. Other Entry Points - none

E. OCLC CCFP Routines Referenced

BIN:BCDR

F. OCLC Procedures Referenced

PUSH
PULL
@ERROR

I. FUNCTIONS

The purpose of this routine is to unpack the call number and all its subfields (stamps, holdings information) and store off call number text that will be used to build the call number sort tag.

Upon entry to FMT:UPCN, the subfields of the call number are unpacked one at a time. Each subfield code in the input field is located in a table of valid subfield codes. If the code is not in the table, an error is declared. When the code is found, its displacement is inserted into a control word and stored as the first word in the unpacked text. If the subfield is holdings information ($d), the holdings indicators from the input record are stored in the next word of the unpacked text. Then the routine FMT:UNPK is used to unpack the subfield data.

Upon return from FMT:UNPK, one of several exits may be taken depending on the return code. If the end of a subfield has been found, the next subfield is processed. If a data link escape has been encountered, an escape control word is inserted and the next subfield is unpacked as above. If an error is found, an error exit is taken. If the end of field or end of record has been encountered by FMT:UNPK, an end of field ontrol word is inserted at the end of the unpacked data. Then the data for the call number sort tag is collected.

The unpacked data is scanned, one word at a time. If the word encountered is a control word, special processing is performed depending on the type of control word. If it is a $a (the call number text), the tag build switch is set and the scan continues. If a stamp is encountered ($b, $c, $p), the call number text switch is turned off and the scan continues.

If holdings information ($d) is found the text pointer is incremented past the holdings indicators, the call number text switch is turned off and the scan continues. If a data link escape control word is found, the tag build switch is inverted and the scan continues. When the end of field control word is encountered, R5 (which contains the numeric portion of the sort tag) is stored in the tag build area and control is returned via R7.

In scanning for the sort tag data, only the call number text is considered. If the text is alpha, it is stored in the tag build area. When the first numeric is encountered, no more alphas are processed. The numerics are accumulated in R5 until a special character (not numeric or alpha) is encountered.

If the special character is part of a numeric fraction, the fraction portion of the tag is built and stored. If a fraction is not present, the scan ends with the first non-numeric encountered, and the sort tag data is complete. Then the numeric portion of the sort tag in R5 is stored in the tag build area. Control is returned through R7.

II. SOFTWARE INTERFACE

A. Linkage

Linkage to FMT:UPCN is accomplished via

```
BAL,R7   FMT:UPCN
```

B. Parameter List Description

Upon entry to FMT:UPCN general registers R1, R2, R3 are set as follows:

R1 - equals BA(call number field in input record)
R2 - equals WA(unpack area)
R3 - equals length of the unpack area

C. Return Codes - none

D. Other Entry Points - none

E. OCLC Subroutines Referenced

FMT:UNPK

F. OCLC Procedures Referenced

@ERROR
SAVE
RESTORE

I. FUNCTIONS

FMT:LKCN links the unpacked main entry paragraph and/or the unpacked title paragraph to the call number depending on the m.c. and t.p. processing indices. If no call number link is specified by the processing indices, control is returned. If the unpacked paragraph is to be linked to the call number, the address of the call number is accessed in P:HEADS and loaded into R1. Then control is returned via R14.

II. SOFTWARE INTERFACE

A. Linkage

The calling sequence is

```
LW,R1     P:ME+1     (processing indices)
BAL,R14   FMT:LKCN
STW,R1    UP:ME+4    (call no. address)
```

B. Parameter List Description

Before linking to FMT:LKCN, R1 should be loaded with the processing indices for the field to be linked.

C. Return Codes

Upon return R1 contains the address of the call number if the field was linked to the call number. If the field was not linked, R1 is unchanged on return.

D. Other Entry Points - none

E. OCLC CCFP Routines Referenced - none

F. OCLC Procedures Referenced - none

I. FUNCTIONS

This routine unpacks the byte text of the input transaction record into word characters. Each word contains all the information needed to generate a symbol or symbols at a given positon on the catalog card, i.e., a character, associated diacritical, and/or underscore. In addition, control bits which ease formatting are carried in each word.

Upon entry to FMT:UNPK, R3 is set up for indexing the target (unpack) area and R9 is set to the address of the unpack routine for the standard character set. Then the first byte of the source is accessed and a branch is taken through R9 to the unpack routine. At the unpack routine, the source byte is used as an index into the character table and the coiresponding word character is loaded into R5. Upon return from the unpack routine, if the character in R5 is zero, the routine UNPK:200 is entered, and an error message is printed for an unprintable symbol. If the word in R5 is not zero, it is interrogated to determine if it is a control word or a diacritical word. If not, the word is text and is stored in the target area. If the word stored was a X'80000040', it is noted with an unprintable symbol message. Then the indices are incremented and the next source byte is unpacked. If the end of the target area is reached before all source bytes are unpacked, an error is declared and control is returned.

If the character word was a control word or diacritical, the routine UNPK:20 is entered and the target word is zeroed. If the word in R5 is an underscore or a diacritical, it is processed at UNPK:40. If it is a control word, it is processed at UNPK:60. If the character word is a diacritical which also contains a text byte, it is stored in the target area as a regular text word and the unpack continues with the next source byte.

At UNPK:40, a diacritical or underscore word is "or'ed" with the target word. The next source byte is accessed and converted to a character word. To be effective, a diacritical or underscore word(s) must be followed by a text word which contains no control information already in the target word and unpack processing continues normally with the next source byte. If the diacritical/underscore is followed by a text word with control information, the second word overlays the first. If two diacritical/underscore words are encountered, the second is "or'ed" with the first and the control information is accumulated. If the diacritical/underscore is followed by a special control word, the target word may be overlaid or "or'ed" with the control word depending on the type of special control word.

At UNPK:60, a control word is processed by first checking for data link escape or end of subfield, field, or record. If any one of the above is present, the return address in R15 is adjusted and control is returned. If the control word is not one mentioned above, a branch is taken to UNPK:80. If the word is a double character word, a branch is taken to UNPK:100 where the table UNPK:DTB is accessed and the double word entry is loaded into R4 and R5. R4 is stored in the current target word, and R5 is stored in the following

word. Normal processing then continues with the next source byte.

If the control word is an escape control word, the next source byte is the character set identifier. The identifier is used to find in the table UNPK:145 the unpack routine for the special character set required. If the identifier is not in the table, an error exit is taken. When found, the routine address is loaded in R9 and processing continues at UNPK:41 (in the diacritical/underscore routine). If the control word is a hyphen, the character word is loaded in R5 and stored in the target. Processing then continues normally. If the control word is a dash, the double word character for a dash is loaded into R4 and R5. Both are stored in the target and normal processing continues.

Depending on the status of the unpack processing at the time control is returned, R15 is incremented with varying values (0-6) and a B *15 is taken.

II. SOFTWARE INTERFACE

A. Linkage

The calling sequence has the following form:

```
BAL,R15     FMT:UNPK
B           RETURN ADDRESS 0
B           RETURN ADDRESS 1
B           RETURN ADDRESS 2
B           RETURN ADDRESS 3
B           RETURN ADDRESS 4
B           RETURN ADDRESS 5
B           RETURN ADDRESS 6
```

where the branch addresses correspond to routines for all possible return situations (see II.D Return Codes).

B. Parameter List Description

Upon entry, the general register should be initialized as follows:

R1 - points to the packed source (BA)
R2 - points to the unpack target area (WA)
R3 - equals the number of unfilled words in the target area.

C. Return Codes

Upon return to the calling program the general registers are set up as follows:

R1 - points to last source byte processed
R2 - points to next available word in target area
R3 - equals the number of remaining unused words in the target area.
R15 - contains the return address incremented by:

0 - Error (target area too small)
1 - Error (syntax error in source)
2 - End of record reached
3 - End of field reached
4 - Delta delimiter found
5 - LB. sign (#) delimiter found
6 - Subfield delimiter found

EXAMPLE OF CALLING SEQUENCE UTILIZING RETURN ADDRESSES:

```
BAL,R15     FMT:UNPK
B           ERROR1
B           ERROR2
MTB,1       ENDSW
B           ENDFIELD
B           DELBR
B           LBBR
B           BETWIXSF
```

D. Other Entry Points - none

E. OCLC CCPP Routines Referenced - none

F. OCLC Procedures Referenced

SAVE
RESTORE
@ERROR

I. FUNCTIONS

This routine builds an unpacked paragraph for the following elements of the catalog card, one at a time:

1. Main entry
2. Uniform or Conventional Title
3. Title Statment
4. Edition Statment
5. Imprint
6. Collation Statement
7. Series Statement (first)
8. Series Statement (others)
9. General Biblio notes
10. Contents Notes
11. Subject Tracing
12. Subject Tracing (special)
13. Other Added Entries
14. Title-type Added Entry
15. Auto-Series Added Entry
16. Series-type Added Entry
17. Heading - Subject Traced
18. Heading - All others
19. Special Notes Field

This is accomplished by unpacking and concatenating all fields for a particular element.

Upon entry to FMT:CC, R1 points to the field indicators in the transaction record, R2 points to the target unpack area (WA). R3 = length of target area. R4 contains the index for the element to a table of processing methods (CC:OPTBL). R5 points to a special subfield type list. If no list is necessary for an element, R5 = 0.

FMT:CC picks up the addresses of the processing routines for this element using R4 and CC:OPTBL. R8 has the address of the routine to process the beginning of the field. R9 has the address of the routine to porcess the following fields; R6 has the address of the between subfield processing routine; R7 has the address of the end of field processing routine.

A branch is taken via R8. At the end of each of the processing routines for beginning of fields, the return is to CC:BF2, which is a branch through R9 to the routine to process the fields. If end of fubfield is encountered during the field routine, the between subfield routine is entered (R6). From the subfield routine, the return is to an interim routine to pick up the next subfield indicator, if present. Then the next field (or subfield) is unpacked. When end of field is encountered, the specified end-of-field routine is entered. After end-of-field processing, the return is via R4.

II. SOFTWARE INTERFACE

A. Linkage

This routine is entered via a

```
BAL,R14   FMT:CC
B         DELETE
```

B. Parameter List Description

Upon entry to FMT:CC the general registers R1-R5 should be set up as indicated:

- R1 - points to the field indicators in the input transaction records.
- R2 - points to target unpack area (word address)
- R3 - equals the length of the target area.
- R4 - equals the index to the processing method.
- R5 - points to a special subfield type list. When one of the types in the list is found, control is given to the exit routine specified in the list. Control must be returned to the routine FMT:CC via an indirect branch on R0. If no list is present, R5=0

C. Return Codes

Upon return from FMT:CC, general registers R1 - R3 and R14 are set as follows. R4 and R5 are unchanged.

- R1 - points to the end of the input record field+1.
- R2 - points to the target and length of the unpacked field.
- R3 - equals the remaining number of bytes in the target area.
- R14 - the value by which the entry value has been incremented:
  - 0 - field should be deleted.
  - 1 - field processing complete.

D. Other Entry Points - none

E. OCLC CCFP Routines Referenced

FMT:UNPK

F. OCLC Procedures Referenced

@ERROR
SAVE
RESTORE

III. DESCRIPTION OF INTERNAL SUBROUTINES

CC:SF This routine saves the subfield code and points R1 to the data to be unpacked. Then a link to FMT:UNPK is made via R15 to unpack the data. Control is returned from FMT:UNPK when an error condition is encountered or the end of a field or subfield is found. The BAL to FMT:CC is followed by a branch table to the various error and end of field or subfield routines.

CC:SFDT If the previous unpack (FMT:UNPK) returned an end of field condition, the between subfield processing address is overlaid in R6 by the end of the field address. Then the subfield code is checked for a lower case 'u'. If the subfield code is 'u', the subfield unpacked is ignored and the next available position is set back to the beginning of the subfield. If the subfield code is not 'u', the ending address of the field or subfield is stored. The end of the unpacked field or subfield is checked for ending pseudo numeric blankds. If any are present they are scanned off backward to the first two nonblank characters. If they are a numeric preceding a hyphen, four numeric blanks are inserted and the end of data address and length are adjusted. If it is a numeric preceding a closing bracket which precedes a hyphen, four numeric blanks are inserted and data address and length adjusted. If the last character before punctuation (- or]) is non-numeric, the return is made via R6.

CC:TEST1 This routine checks CC:SAVE for a possible subfield type list. If CC:SAVE is zero, a branch is taken to CC:SF to process the next field or subfield. Otherwise the next subfield indicator is accessed via R1 and set up in R5.

CC:TEST2 Here the subfield indicator set up by CC:TEST1 in R5 is compared to the next one in the subfield processing table. If either is zero, a branch is taken to CC:SF to process the next subfield. If the subfield indicators are not equal, the next entry in the table is interrogated. If they are equal, the address of the exit routine is picked up, and the special processing exit for that subfield is taken. Upon return from the special exit, a branch is taken to CC:SF.

The following routines may be utilized in pre-field processing.

CC:BF10 There is no special processing in this routine. The beginning field processing routine is entered immediately via R9.

CC:BF11 This routine will put an arabic sequence number at the beginning of the paragraph. Then control is transferred via R9 to the beginning field processing routine.

CC:BF12,13,14,15 This routine puts a roman sequence number at the beginning of the paragraph. A branch is then made via R9 to the beginning field processing routine.

The following routines may be used in po-cessing before the first field.

CC:BF20 There is no special processing in this routine. The first field or subfield address is set up in R1 and control goes to CC:TEST1.

CC:BF21 This routine inserts a left parenthesis before the first field. Field length and address are incremented to include the extra character. Then control falls through to CC:BF20.

CC:BF22 A left bracket is inserted before the first field. Field length and pointers are adjusted to include the extra character.

CC:BF23 This routine inserts a no-break period and a blank before the first field. Field length and address are adjusted to include the two new characters. A branch is then taken tp CC:BF:20.

CC:BF24 A no-break period and mark are inserted before the first field. Field length and pointers are adjusted to include the characters. A branch is then taken to CC:BF20.

CC:BF25 This routine inserts the words "contents.--" or "partial contents.--" before the first field depending on the field indicator. If the indicator is 1 (one), the field is deleted. If the indicator is 0, "contents.--" is used. If the indicator is greater than 1, "partial contents.--" is used. The length and address of the field are adjusted and the appropriate words are moved in. Control goes to CC:BF20.

CC:BF26 The word 'TITLE:' is inserted before the first field. The length and address of the word is set up and CC:MOVE2 moves the data, adjusts the field address, and transfers control to CC:BF20.

CC:BF27 The word 'SERIES:' is inserted before the first field. The length and address of the word is set up, and CC:MOVE2 moves the data, adjusts the address and length and transfers control to CC:BF20.

The following routines may be utilized inbetween field processing.

CC:SF0 CC:SF0 will insert one blank between fields. The address of the next available output position is adjusted. Then a branch is made in CC:TEST1.

CC:SF1 The routine will insert two blanks between fields and adjust the target address. A branch is then taken to CC:TEST1.

CC:SF2 Two fields are separated by a pseudo-dash. The target address is adjusted and a branch is made to CC:TEST1.

CC:SF3 This routine checks to see if the subfield just processed was an X,Y, or Z subfield. If yes, a psuedo dash is inserted at CC:SF2. If not, a blank is inserted at CC:SF0.

The following routines may be used in end of field processing.

CC:EF0 This routine increments R14, the return register, by one to indicate processing is complete. If the switch to save heading information is set, the address of the heading information is stored in the table P:HEADS before returning. If the switch is not set, the registers are restored and control is returned via R14.

CC:EF1 A right parenthesis is added to the end of the field. Then control falls through to CC:EF0.

CC:EF2 A right bracket is added as in CC:EF1.

I. FUNCTIONS

FMT:HSV saves the pointers to the packed field in the heading pointer table if necessary. Upon entry the second indicator for the field is checked. If it is one (1), R6 is restored and control is returned through R14. If the second indicator is not one, the location of the packed field in the input record is saved in the heading pointer table. The processing indices for the field are also saved. These two words in the table are followed by two words of -1. Then the table pointer is incremented by 4 to the next available position and a -1 is stored there to mark the end of the table. The updated table pointer is stored in HSV:P. Then R6 is restored and control is returned through R4. If the heading pointer table is full upon entry, an error message is printed and control is returned to FMT:INIT.

II. SOFTWARE INTERFACE

A. Linkage

Control is passed to FMT:HSV via a

```
BAL,14   FMT:HSV
```

B. Parameter List Description

Upon entry to FMT:HSV

C. Return Codes - none

D. Other Entry Points - none

E. OCLC CCFP Routines Referenced - none

F. OCLC Procedures Referenced -

@ERROR

I. FUNCTIONS

The purpose of this routine is to build the unpacked continuation policy when a continuation card is necessary. The continuation is built once per set of cards. Upon entry to FMT:UPCP the UPCP internal processing switches are set to zero. The main entry UPP pointer table is then interrogated. If a main entry is not present, the title is used as the main entry and is moved to the continuation policy area. Then further title processing is skipped and the rest of the continuation policy is built at UPCP:TRLR. If a main entry is present, its UPP is moved to the continuation policy area. Then the title UPP pointer table is checked. If the author was the main entry, the title UPP is inserted in the continuation policy area. If a uniform title is present, it is used as the title.

To insert the title or uniform title UPP into the continuation policy, the width of the line is calculated using the title first indention. If the entrie title UPP will fit on one line, title processing is begun. Otherwise UPCP:TRUNC is initialized with the line length to show that the UPP will be truncated, and the end of the UPP is reset to the end of the line. To move the title to the continuation policy area, the character words are picked up from the UPP and stored in the area from left to right. After each word is stored, it isinterrogated to determine if it is a break character. If not, the line displacement is incremented and the next character word is stored.

If the character stored is a left break and it is a blank, the line displacement is stored in UPCP:BRK. If the preceding character is not right break punctuation, the line displacement after the blank is stored in UPCP:LOG. Then the line displacement is incremented and processing continues on the remaining characters. If the character stored is a left break, but not a blank, it is checked to see if it is a left bracket. If so, the switch UPCP:BRAC is incremented by one. Then processing continues on the remaining characters.

If the character stored is a right break, it is checked to see if it is a right bracket. If so, the switch UPCP:BRAC is decremented by one. Then the remaining characters are processed.

When the end of the line is encountered if the UPP was not truncated, processing for the remainder of the continuation policy is begun at UPCP:TRLR. Otherwise the break point of the line is determined. If the last character is a blank and no other break puncuation was found, or if no blanks or punctuation were found, the entire line is used. If the last character is non-blank and no other break punctuation was encountered, the line is broken at the last blank. Otherwise the line is broken at the punctuation indicated by UPCP:LOG. Then the temp area pointers are adjusted.

At UPCP:TRLR, a dot ellipsis '...' is inserted. If a left bracket and no right was found in the title, a right bracket is inserted following the dot ellipsis. Then the input record map is checked to see if imprint data is present. If not, a branch is taken to UPCP:T60 to end the continuation policy. If imprint data is present, it's UPP is built using the field concatenation routine FMT:CC. Then the imprint UPP is moved to the continuation policy area. Whenever a left bracket is encountered, R1 is incremented by one. When a right bracket is encountered, R1 is decremented by one. At the end of the imprint UPP, R1 is checked. If it is zero (either no brackets or a set of brackets), a branch is taken to UPCP:T60 to process the end of the continuation policy. If only a left bracket was present and the last character was punctuation, a right bracket is stored in place of the punctuation. If no punctuation ends the UPP, a right bracket is stored following the UPP and the temp area pointers are adjusted. Then control goes to UPCP:T60. If only a right bracket was found, the entire imprint UPP is shifted to the right one position in the continuation policy area. Then a left bracket is inserted at the beginning of the imprint UPP.

At UPCP:T60 the words '(CARD N)' are inserted at the end of the continuation policy. The end address of the UPCP is stored in the UPCP pointer table. Then control is returned to the calling routine.

II. SOFTWARE INTERFACE

A. Linkage

BAL,R7 FMT:UPCP

B. Parameter List Description

Upon entry to FMT:UPCP R2 must point to the continuation policy temp area and R3 must equal the CP temp area remaining length.

C. Return Codes - none

D. Other Entry Points - none

E. OCLC CCFP Routines REferenced

FMT:CC

F. OCLC Procedures Referenced

PUSH
PULL

I. FUNCTIONS

FMT:CAP will capitalize a text sting. It checks each character word in the string. If the character word is alpha and lower case, it is capitalized; otherwise the next word is interrogated. Return is via R7.

II. SOFTWARE INTERFACE

A. Linkage

```
BAL,R7    FMT:CAP
```

B. Parameter List Description

Upon entry to FMT:CAP general registers R4 and R6 should be set up as follows:

R4 - points to beginning of unpacked text string.
R6 - points to (end of unpacked test string) +1.

C. Return Codes - none

D. Other Entry Points - none

E. OCLC CCFP Routines Referenced - none

F. OCLC Procedures Referenced - none.

I. FUNCTIONS

FMT:DCAP decapitalizes an unpacked text string. It checks each character word in the string. If the word is not a capital it is ignored; otherwise it is decapitalized. Return is via R7.

II. SOFTWARE INTERFACE

A. Linkage

```
BAL,R7    FMT:DCAP
```

B. Parameter List Description

Upon entry to FMT:DCAP R4 and R6 should be set as follows:

R4 - points to beginning of unpacked text.
R6 - points to (end of unpacked text) +1.

C. Return Codes - none

D. Other Entry Points - none

E. OCLC CCFP Routines Referenced - none

F. OCLC Procedures Referenced - none

I. FUNCTIONS

FMT:SP scans the unpacked paragraph into lines, breaking at valid punctuation or blanks. At its initial entry point, FMT:SP1, the first indention value for the paragraph is used. At SP:05 the maximum line width is computed using the first indention. If the maximum line length exceeds the remaining paragraph length, then no scan is necessary and control goes to SP:200 where the return address is incremented by one and control is returned via R14. If more than one line is left in the paragraph, the maximum number of character words in the line are scanned from right to left until a valid character is found with which to end the line. The line is broken at a blank or a punctuation character.

When a right break character is encountered (break should come after character such as ']', '}', '>', '%', ':'), the character to its right is checked. If it is not another right break character, the line is ended after the first character. Otherwise, the scan continues to the left until a break other than a right break character is found.

When a break character is found which is both a left and right break character ('=', '+', '-', '&'), the line is ended after the character. When a left break character is encountered (break should come before character as '[', '(', '<', '$', '@'), the character to its left is checked. If it is not another left break, the line is ended before the first character. If the next left character is a left break, the scan continues to the left until a break character other than a left break character is found. Then the line is broken.

If no break character is found within the maximum limits of the line, an error is declared and control is returned via R14. When a valid break character is found, control goes to SP:60 where the current line count passed by the calling routine is incremented by one. If the resulting line number is equal to the stop line number passed by the calling routine control is returned by incrementing the return address by two to indicate the end of a line and branching via R14. If the line count is not equal to the stop line number, control falls through to SP:80 where the next characters in the paragraph are scanned until a non-blank is found. Then the remaining paragraph length is adjusted, the second indention value is set up, and abranch is taken to SP:05 to set up the next line length.

At its alternate entry point, FMT:SP2, the routine sets up the remaining characters in the paragraph, clears the line length and branches to SP:80 to scan for the start of the next line and set up the second indention.

II. SOFTWARE INTERFACE

A. Linkage

```
BAL,R7   FMT:SP1
   or
BAL,R7   FMT:SP2
```

B. Parameter List Description

Upon entry to either FMT:SP1 or FMT:SP2 the general register must be set as follows:

R6 - points to the list of indention values for the paragraph

R8 - points to the unpacked text within the paragraph.

R9 - contains the line number assigned to the first scan line.

R10 - contains the stop line number.

R12 - points to the end of the paragraph +1.

R11 is destroyed by this routine.

C. Return Codes

If the end of a line is found, the return address is incremented by two. If the end of a paragraph is found, the return address is incremented by one. If an error is declared, the return address is not incremented.

Upon return the general registers are set by FMT:SP1 and FMT:SP2 as follows:

R6 - still points to list of indention values.

R8 - points to the beginning of the last line of unpacked text.

R9 - contains the stop line number +1, or the last line of the paragraph +1.

R10 - still contains the stop line number.

R11 - points to the end of the last line of unpacked text +1.

R12 - points to the end of the paragraph +1.

D. Other Entry Points

FMT:SP1
FMT:SP2

E. OCLC CCFP Routines Referenced - none

F. OCLC Procedures Referenced

SAVE
RESTORE

I. FUNCTIONS

The purpose of this routine is to repack or seppack each unpacked paragraph into lines on the catalog card. The routine keeps track of how many lines have been used on the card or cards so far and where the first line of text should begin on extension cards. It adjusts the lines on the card around the rod hole and provides for continuation cards. FMT:SPK is entered at least once for each unpacked paragraph to be formatted.

Upon entry to FMT:SPK three items of information must be present from the calling routine. R5 must point to the unpacked paragraph's table entry, R6 must contain the paragraphs indention index in the table INDT:VAL, and the switch SW:SPK may be set to a value less than zero to indicate forced reseppacking. Several reserved words are kept as running totals during the processing of a catalog card: CUR:LN contains the current line number of the card being formatted, CUR:FLTX contains the line number of the first line of text for extension cards, PREV:CLS contains the number of lines so far on this card (including lines on extension cards).

FMT:SPK first sets R6 with the address of the set of indentions for this paragraph. Then the type of seppacking to be performed is determined. The address of the temp area where the seppacked paragraph (SPP) will go is interrogated in the unpacked paragraph (UPP) pointer table. If the address is present, re-seppacking is being requested for the UPP and control goes to SPK:600. If the address is set to -1, this is the first entry for this UPP to be seppacked and control is transferred to SPK:620. If the SPP address in the UPP pointer table is zero and the switch SW:SPK has a negative value, re-seppacking is being forced for this UPP and probably for the entire card and control goes to SPK:610. If the address in the UPP pointer table is zero and SW:SPK is not negative, the address of the associated unpacked call number is checked. If a call number is associated with this paragraph and it is not the primary call number, the call number and its associated UPP will be re-seppacked beginning at SPK:610. If it is the primary call number and text extends into the call number area, the call number and the text are re-seppacked beginning at SPK:610. If no text extends into the primary call number area, the call number is reseppacked. If no call number is present but the text extends into the call number area, reseppacking must be performed starting at SPK:610. If the SPP address is zero, SW:SPK is greater than or equal to zero, the test does not extend into the call number area, no call number other than the primary call number is present, and the old SPP has not been wiped out, then control goes to SPK:410 where an attempt is made to adjust the SPP without reseppacking.

At SPK:410 checks are made to determine if the SPP can be adjusted in the temp area. If the new SPP will start on the same line as the old SPP and the old SPP does not span cards, the SPP can be adjusted. Control goes to SPK:430 to adjust the old SPP in the temp area. If the old SPP spans cards and the continuation policy is not yet built, the SPP must be reseppacked. If the SPP spans cards, the continuation policy is built,

and the new SPP will have the same span line number as the old, then the SPP can be adjusted; otherwise it must be reseppacked. If the old SPP both spans cards and intersects the rod hole of the card, reseppacking must be performed. If the old SPP is on one card but intersects the rod hole, the desired line number of the new SPP is checked to see if the new SPP will clear the rod hole. If not, the old SPP status is checked to see if the SPP has a short last line. If so, the SPP line count is decremented by one; and a check is again made to see if the new SPP will clear the rod hole. If so, the SPP is adjusted. Otherwise it must be reseppacked. If no call number is associated with the old SPP, the SPP is adjusted to the desired line number, control fields in the SPP leader are updated as well as CUR:LN and PREV:CLS, and control is returned to the calling routine. If there is a call number associated with the old SPP, it is reseppacked. If the reseppacked call number has an extra length line which interferes with the desired location of the SPP, the entire SPP must be reseppacked. If the reseppacked call number does not interfer with the desired location of the new SPP, the old SPP is adjusted as above and control is returned.

At SPK:600 a check is made to see if this is the first time through for this UPP. If so, a branch is taken to SPK:620 to set up for seppacking. If not, at SPK:610 an attempt is made to adjust the temp area pointer so that reseppacking will be performed in the same area that the old SPP occupied. If the temp area pointer is now pointing past the old SPP, then the old area cannot be used and a branch is taken to SPK:620. Otherwise the temp area pointer is adjusted and reseppacking is done in the old SPP area.

At SPK:620 words 2, 4, and 5 of the SPP leader are initialized. Then at SPK:630 a check is made to see if there is a call number associated with the UPP. If so, the call number is seppacked at this time using the routine FMT:SPCN. If an error occurs, recovery is attempted in the routine SPK:ERCN. Whether or not a call number is associated with the UPP, if the text intersects the call number area, the current line number is adjusted until it clears the call number. If the beginning line number or the adjusted beginning line number is the bottom card line, a continuation is necessary. The associated call number's beginning, ending, and extra length line numbers are replaced with those of the primary call number. The associated call number will occupy the primary call number position on the continuation card. The number of lines for this set of cards is incremented by 23 (maximum lines per card). Then the continuation policy is seppacked with its text beginning on the CUR:FLTX line number. After the continuation policy is seppacked by the routine SPK:CP, the associated call number must be reseppacked on the new card. For this purpose a branch is taken to SPK:630. If a usable continuation policy is not formulated after two entries to SPK:630 an error is declared nad processing for the current catalog card request is terminated.

If no continuation was required after seppacking or if the continuation policy was seppacked successfully, SPK:636 is entered. Here the beginning address of the SPP is stored in the UPP pointer table. Parameters are then set up for the UPP scan routine for the first line of the paragraph FMT:SP1. FMT:SP1

is used to find the first line of the paragraph. If no break point is found in the line, an error is declared and processing is terminated for this card request. If the end of the UPP was encountered, the switch SW:ENDP is set to a positive value. If a line was found normally, seppacking is begun.

At SPK:640 the first indention value is checked. If a first indention is not begin used, a branch is taken to SPK:642; otherwise, the tab value control byte (X'FD') is inserted followed by the tab value. At SPK:642 if the current line number is within the rod hole limits, the branch SPK:BIR is set to SPK:656 and the length of the first half of the rod hole line is calculated.

If the line number is outside the rod hole limits, SPK:BIR is set to SPK:658. At SPK:650 the unpacked character word for the next print position is interrogated. If it contains no control information, a branch is taken to SPK:654. If the control information is a diacritical, control is transferred to SPK:651. If the control is an underscore, the branch is to SPK:652. If the control information is a tab value, a branch is taken to SPK:650T. Otherwise the control was a paragraph delimiter within the UPP. In this case R6 is incremented to the second set of indentions for this SPP. An end of line charcter (X'FE') is stored after the last character seppacked. If the UPP pointer is still at the beginning of the line, a check is made to see if the previous two characters were a tab. If so, the tab is wiped out. The first unpacked character word is skipped, the line count is decremented and another attempt is made to scan the line at SPK:BSCAN. If no tab value was present, the line count and character word pointer are adjusted as above and control goes to SPK:BSCAN. If any characters were seppacked, the next UPP word is pointed to and control goes to SPK:BSCAN.

At SPK:650T, the tab value control byte is inserted and a branch is taken to SPK:654 where the tab value will be inserted. At SPK:651 the diacritical control byte (X'FC') is inserted followed by the diacritical from the character word, then control falls through to SPK:652. At SPK:652 the character word is checked again to see if it contains an underscore. If not, a branch is taken to SPK:654, otherwise, the underscore bit is turned off and control goes to SPK:651 to insert a diacritical control byte and the underscore.

At SPK:654 the live character is stored in the SPP area, the characters per line count is incremented and the branch SPK:BIR is taken (to SPK:658 unless the line intersects the rod hole). If the entire line has been seppacked, the branch is to SPK:660. At SPK:656 the first half line character count for the rod hole line is decremented. If the first half of the line is not full, seppacking continues. If the first half of the rod hole line is complete, the branch SPK:BIR is reset from SPK:656 to SPK:658 for the remainder of the line. The SPP status is set to indicate

that this SPP intersects the rod hole. Then a tab is set to space across the rod hole and seppacking continues at SPK:658. At SPK:658 the temp area remaining length is decremented. If the area is full, an error is declared and processing for this card request is terminated. Otherwise the next character is seppacked at SPK:650.

At SPK:660 a check is made to see if the end of the card has been reached. If not, a branch is taken to SPK:665, otherwise a continuation card may be needed. The total lines for this set of cards, PREV:CLS, is incremented by 23. The call number beginning, ending, and extra length line numbers are reset to those of the primary call number. If the line number of the first line of text for continuation cards (CUR:FLTX) is greater than zero, the continuation policy has already been seppacked. If CUR:FLTX is zero, the last paragraph switch is checked. If this is the last paragraph and the last line of the paragraph has been seppacked (SE:ENDP not zero), no continuation is necessary and a branch is taken to SPK:670. Otherwise the routine SPK:CP is used to seppack the continuation policy. After the continuation policy is seppacked, if this is the end of a paragraph, a branch is taken to SPK:670. If this is not the end of a paragraph, the second indention is checked. If no second indention is present, a branch is taken to SPK:664; otherwise, CUR:FLTX is ompared to the call number ending line number (CN:ELN). If CUR:FLTX is less than or equal to CN:ELN, then CN:ELN is incremented by one; and this value is stored as both CN:ELN and the current line number (CUR:LN). If CUR:FLTX is greater than CN:ELN, CUR:FLTX is stored in CN:ELN and CUR:LN. Then the end of paragraph control byte (X'FF') is stored at the end of the card. The current lines on the continuation card are added to PREV:CLS, and the line number of the first line on the new card is stored in the next position in the SPP area. Finally, a branch is taken to SPK:667.

At SPK:664, the SPP status is set to indicate that the paragraph is spanned. Then end of line processing is entered. At SPK:665 if the end of paragraph switch is not zero, the end of a paragraph was found and a branch is taken to SPK:670. Otherwise end of line processing is initiated. Trailing blanks are scanned off the line and the end of line control byte (X'FE') is inserted following the last non-blank character. Then the paragraph scan routine is entered at its second entry point, FMT:SP2, to scan for the next line of the paragraph. Upon return from FMT:SP2, the next indention value is set up and control goes to SPK:640 to seppack the line.

SPK:670 is the end of paragraph routine. The end of paragraph control byte (X'FF') is stored after the last seppacked line. Then the last line of the SPP is checked to see if it is a short line (would fit on the first half of the rod hole line). If so, the SPP status is set; otherwise, the remainder of the SPP leader is built. R2 is adjusted to the next available position in the temp area. The length of the SPP is calculated and stored in the SPP leader in Word3 with CUR:FLTX which indicates whether or not a continuation policy has been seppacked for this set of cards. CUR:LN is stored in Word0 of the SPP leader with the entry CUR:LN. PREV:CLS is stored in WORD1 of the SPP leader with the entry PREV:CLS. If this was the first time the UPP was

seppacked, the temp area pointer is incremented by ten words and a branch is taken to SPK:695. Otherwise the old next SPP area address is interrogated in the SPP leader. If it is less than the current temp area address, a branch is taken to SPK:695 where the old address is used as the next SPP area address. Otherwise, the temp area pointer is readjusted to the old next SPP address from the leader.

At SPK:695 the SPP status is "OR'ED" with the next SPP area address and the resulting word is stored in WORD5 of the SPP leader. Then at SPK:700, the switch SPK:RENT is checked to see if the routine was recursively re-entered from SPK:RRR to continue seppacking. If not, the return is via R7 to the original calling routine.

II. SOFTWARE INTERFACE

A. Linkage

BAL,R7 FMT:SPK

B. Parameter List Description

Upon entry to FMT:SPK R5 and R6 should be set as follows:

R5 - points to the unpacked paragraph's table entry.
R6 - equals the indention index into the table INDT:VAL for this paragraph.

In addition the switch SW:SPK may be set to a negative value to indicate that reseppacking is being forced.

C. Return Codes - none

D. Other Entry Points - none

E. OCLC CCFP Routines Referenced

SPK:CP
SPK:ERCN
FMT:SPCN
FMT:SP1,SP2

F. OCLC Procedures Referenced

SAVE
RESTORE

III. SPECIAL STORAGE AREAS, SWITCHES, AND TABLES

A. Special Storage

Seppacked Paragraph - the finished seppacked paragraph is laid out as follows:

SPP LEADER:

HW-0 CUR:LN upon exit
HW-1 CUR:LN upon entry
HW-2 PREV:CLS upon exit
HW-3 PREV:CLS upon entry
HW-4 (reserved for call number positioning)
HW-5 (reserved for call number positioning)
HW-6 current CUR:FLTX
HW-7 byte length of SPP (excluding the leader).
HW-4 (reserved for call number positioning)
HW-10 status of the SPP, meaning if BIT=1:

BIT-0 SPP spans two or more cards
BIT-1 SPP intersects the rod hole
BIT-2 SPP ends with a short line

HW-11 beginning of temp area which next paragraph may use.

SPP TEXT:

The first byte of each SPP contains the absolute line number on which the paragraph begins. Line number one is the first line of the first card. The remaining bytes are either text bytes or control bytes.

SPP Control BYTES:

X'FF' end of SPP or beginning of a call number paragraph
X'FE' end of a card line
X'FD' tab value follows
X'FC' diacritical follows

I. FUNCTIONS

SPK:CP sets up parameters to allow the continuation policy for a set of cards to be seppacked by recursively reentering FMT:SPK. Upon entry the switch SPK:RENT is checked. If SPK:CP has been entered twice consecutively, an error is declared and processing is terminated for this card request. Otherwise the registers from FMT:SPK are pushed into the stack and the seppack control information is saved. Then CUR:LN is set to the line number of the first card's first line. The continuation policy's temp area pointers (saved in CP:TEMP) are loaded into R2 and R3 for FMT:SPK.

If the continuation policy has not yet been built (unpacked), a branch is taken to the routine FMT:UPCP via R7. Then the address of the continuation policy's UPP ~~table entry is~~ set up in R5. The main entry indentions are set up in R6. If there is no main entry, the indentions for the title as main entry are used. Finally a branch is taken to SPK:001 within FMT:SPK to seppack the continuation policy.

When the continuation policy seppacking is complete, the return from FMT:SPK is to SPK:RRR where the reentrant switch is reset. The registers and control information for FMT:SPK are restored. If the CUR:LN after the continuation policy is in the rod hole area, an error is declared and processing is terminated for this card request. Otherwise CUR:FLTX is initialized to the CUR:LN and return is via R7 to FMT:SPK.

II. SOFTWARE INTERFACE

A. Linkage

BAL,R7 SPK:CP

B. Parameter List Description

CUR:FCFL must be initialized with the line number of the first line of the first card.

C. Return Codes - none

D. Other Entry Points

SPK:RRR

E. OCLC CCFP Routines Referenced

FMT:UPCP

F. OCLC Procedures Referenced

PUSH
PULL
SAVE
RESTORE

I. FUNCTIONS

SPK:ERCN attempts to recover from an error situation in FMT:SPK in which the call number is too large to fit in the remaining space on the card. If this routine has been entered twice for the current paragraph, a branch is taken to SPK:ERC1. Otherwise the primary call number beginning line number is checked. If the primary call number is the one with the error, control goes to SPK:ERC1. If not, the primary call number's beginning, ending, and extra length line numbers are stored as those of the current call number. If the continuation policy is already seppacked (CUR:FLTX is greater than zero), control is returned via a B -2,R1 to initiate reseppacking of the call number. If CUR:FLTX is less than or equal to zero, the reoutine SPK:CP is entered to seppack the continuation policy. Then control is returned to FMT:SPK to retry call number seppacking.

At SPK:ERC1, the switch MINOR:SW is checked to see if this is the first error for this set of cards. If so, a branch is taken to RECOVER to attempt to restart processing for this set of cards. If more than one error has occurred, a call number error message is printed and call number processing is skipped in FMT:SPK for the current paragraph.

II. SOFTWARE INTERFACE

A. Linkage

```
BAL,R1     SPK:ERCN
```

B. Parameter List Description - none

C. Return Codes - none

D. Other Entry Points - none

E. OCLC CCFP Routines Referenced

SPK:CP

F. OCLC Procedures Referenced

@ERROR

I. FUNCTIONS

The purpose of FMT:SPCN is to seppack the unpacked call number paragraph onto the catalog card. Upon entry to FMT:SPCN from FMT:SPK, R1 points to the unpacked call number, R2 points to the associated seppacked paragraph (SPP) leader, and CUR:LN contains the current card line number. After setting up the temp area address for the seppacked call number (SPCN) and storing it in the SPP leader, the SPCN text identification control byte (X'FF') is stored as the first byte of the SPCN.

If the call number to be seppacked is the primary call number, the call number first line is raised above the current line by one, if possible, or to the highest permissable line number. Then control goes to SCN:30. If this is a secondary call number and there is a second indention for the associated paragraph, the last call number ending line number (CN:ELN) is incremented by one. If no indention is present, the last CN:ELN is checked to make sure that the current call number will begin on the first line below the last call number on the card. If not, the current call number beginning line number is set to CN:ELN+1. If this line number exceeds the permissable card limits, an error is returned to the calling routine. Otherwise CUR:LN is reset to force the associated paragraph to start below the first call number line.

At SCN:30, the current call number beginning line number is saved in CN:BLN. The beginning card line number is inserted in the next temp area position, and the call number line count and line width are reset. Then the first character word of the unpacked call number text is interrogated at SCN:100. If any of the first four bits are set, or if the word is not a control word, control goes to SCN:110. Otherwise a control word is assumed and an end of line control byte is inserted in the temp area. If the control word is a data link escape, the next word is interrogated. If the current line is oversize, the line number is saved. Then the special handling routine for this control word is entered: $a-SCN:A, $b-SCN:B, $c-SCN:C, $d-DCN:D, $p-SCN:P, EOF-SCN:EOF.

At SCN:110, a check is made to see if the word is a diacritical. If not, a branch is taken to SCN:120. If so, the diacritical control byte (X'FC') is stored in the next target position followed by the diacritical. Then control falls through to SCN:120 where the live text byte is inserted in the target area. The source and target pointers are adjusted and control returns to SCN:100 to pick up the next character word.

SCN:A increments the call number line count and returns control to SCN:100 to pick up the next character word. SCN:B checks to see if a non-PDT stamp-type 1 is allowed by the PDT. If not, the $b subfield is skipped by branching to SCN:SCAN. Otherwise, control goes to SCN:A. SCN:C checks to see if a non-PDT stamp-type 2 is allowed by the PDT. If not, the $c subfield is skipped by branching to SCN:SCAN. Otherwise control goes to SCN:A. SCN:D sets up the holdings indicators in R6 and uses the routine FMT:SPTS to see if

holdings are allowed. If not, the $d subfield is skipped by branching to SCN:SCAN. Otherwise control goes to SCN:A.

SCN:P checks the PDT to see if a PDT stamp is present. If not, a branch is taken to SCN:SCAN to find the next subfield. Otherwise a new call number line is begun and the PDT stamp is moved byte by byte from the PDT to the target area. After the PDT stamp has been moved to the target a new line character is inserted, CN:XLN1 is updated if the PDT stamp is greater than the maximum call number width, and the next subfield is scanned for by branching to SCN:SCAN.

At SCN:EOF the end of paragraph control byte is inserted at the end of the call number text. The call number ending line number is calculated and if the call number has exceeded its area, an error is declared. Otherwise the ending line number incremented by one is saved in CN:ELN. If there is an extra long call number line, its line number incremented by one is saved in CN:XLN. If this is the primary call number, the beginning, ending and extra length line numbers are saved at CN:BLNP, CN:ELNP, and CN:XLNP respectively. The call number's last line number is stored in the SPP leader along with the SPAN length. Then the call number temp area pointers are adjusted and saved, registers are restored, and control is returned via a B 1,R7.

II. SOFTWARE INTERFACE

A. Linkage

```
BAL,R7  FMT:SPCN
```

B. Parameter List Description
Upon entry to FMT:SPCN R1 and R5 must be set up as follows:

R1 - WA(unpacked call number)
R5 - WA(associated SPP leader)

In addition CUR:LN must contain the current card's current line number and PREV:CLS must contain the current card set's current line number.

C. Return Codes

If an error is declared, return is via a B 0,R7.

If seppacking was accomplished normally, return is via a B 1,R7.

D. Other Entry Points - none

E. OCLC CCFP Routines Referenced

FMT:SPTS

F. OCLC Procedures Referenced

SAVE
RESTORE
@ERROR

I. FUNCTIONS

The purpose of FMT:SPSP is to check the special notes indicators; and if special notes are allowed FMT:SPSP uses the routine FMT:SPK to seppack the special notes paragraphs. Upon entry to FMT:SPSP R5 must be pointing to the list of pointer table entries for the current type of special notes UPPs (399, 499, 599, or 899). The high order bit of the SPP address in the UPP table entry is set to one. Then the field indicators are set up and the routine FMT:SPTS is used to validate the indicators. If the indicators are false, the paragraph is skipped and the next special notes UPP table is checked. If the field indicators are valid, the indention index is set up in R6, and the routine FMT:SPK is used to seppack the special notes paragraph.

After the paragraph is seppacked, the next set of UPP pointers is checked. If the first word is less than one, there are no more special notes paragraphs and control is returned to the calling program. Otherwise processing for the next special notes paragraph is as described previously.

II. SOFTWARE INTERFACE

A. Linkage

```
BAL,R7   FMT:SPSP
```

B. Parameter List Description

Upon entry to FMT:SPSP, R5 must be pointing to the set of UPP pointer tables for the special notes paragraphs to be formatted.

C. Return Codes - none

D. Other Entry Points - none

E. OCLC CCFP Subroutines Referenced

FMT:SPTS
FMT:SPK

F. OCLC Procedures Referenced

SAVE
RESTORE

I. FUNCTIONS

The purpose of FMT:INSN is to act as a driver for inserting the special notes paragraphs in the output record buffer. Upon entry to FMT:INSN, the address of the seppacked paragraph (SPP) leader is picked up from Word 3 of the unpacked paragraph (UPP) pointer table. If the address is -1, the next special notes UPP table entry is checked. Otherwise the routine FMT:INSP is used to move the SPP into the next position in the output buffer. Upon return from FMT:INSP, the next UPP table entry is checked. If the first word is -1, there are no more special notes UPP's and control is returned to the calling routine. Otherwise the next special notes paragraph is moved to the output buffer.

II. SOFTWARE INTERFACE

A. Linkage

```
BAL,R7    FMT:INSN
```

B. Parameter List Description

Upon entry to FMT:INSN, R5 must point to the first special notes UPP pointer table entry.

C. Return Codes - none

D. Other Entry Points - none

E. OCLC CCFP Routines Referenced

FMT:INSP

F. OCLC Procedures Referenced

```
SAVE
RESTORE
```

I. FUNCTIONS

FMT:SPTS tests special indicators in the input fields against specifications in the PDT to determine if partial holdings, multiple holdings or special notes are to be formatted on the catalog card.

If the PDT indicator for partial holdings, HD:HLD1 is set and the partial holdings off bit is not set in the input indicators, partial holdings will be formatted. If the PDT indicator for multiple holdings, HD:HLD2, is set and the multiple holdings off bit is not set, the holdings will be formatted. Otherwise, return to the calling routine is via a B 0,R7 to indicate that holdings indicators are false.

If the shelf list card bit is set in the special notes indicators, the CRD:TYPE indicator in the PDT must be one. If the main entry card bit is set in the special notes indicators, CRD:TYPE must be two. If the other entry card bit is set in the special notes indicators, CRD:TYPE must be less than three. If any of the tests fail, return is via a B 0,R7. Otherwise, control is returned via a B 1.R7.

II. SOFTWARE INTERFACE

A. Linkage

```
BAL,R7    FMT:SPTS
```

B. Parameter List Description

Upon entry to FMT:SPTS, R6 must contain the indicators to be checked.

C. Return Codes

If a test fails return address is through R7.
If all tests are successful, return is via a B 1,R7.

D. Other Entry Points - none

E. OCLC CCFP Routines Referenced - none

F. OCLC Procedures Referenced - none

I. FUNCTIONS

FMT:TAG builds the sort tag for a field from the unpacked text. Each unpacked character word is interrogated. If the character is significant, it is stored in the tag build area. Abbreviated words are replaced with their unabbreviated form. Articles and other punctuation are ignored. When the tag build area is full, the tag bytes are compressed into 6-bit representation and stored back in the tag build area.

Upon entry to FMT:TAG, the source string length is calculated. A character word is interrogated at TG:LOOP. If it is punctuation, control goes to TG:PUNC. If not, it is stored as part of the current tag word (WD). If the character is numeric, a branch is taken to TG:NUM. Otherwise, it is an alpha. If an Umlaut is present with the alpha, the alpha is stored in the tag build area followed by an 'E' to denote an Umlaut in the tag. If no Umlaut is present, the alpha is stored in the tag area and the next source character word is interrogated. At TG:NUM the character byte is stored in the next tag position. Then at TG:COND the character is stored at TG:PRVCH as the previous character. At TG:ELOOP the source string displacement is incremented and control goes back to TG:LOOP to interrogate the next character.

At TG:PUNC the punctuation character byte is interrogated. If it is a blank, a branch is taken to TG:BLNK. If it is a hyphen and the previous character was a hyphen, the article switch TG:ARTSW is set to the value of the hyphen and a branch is taken to TG:ABBRV+3. If the punctuation is neither a blank nor a hyphen, it is checked to see if it is a single quote. If so, and if the previous character was 'M', the character byte is changed to 'c' and a branch is taken into TG:LOOP to store it in the tag area. Otherwise control goes to TG:COND.

At TG:BLNK, if the previous character was also a blank, this character is ignored and a branch is taken to TG:ELOOP to loop to get the next source word. Otherwise, if an article has already been tested for, a branch is taken to TG:ABBRV. If no article test was made but the previous character was a '.', a branch is taken to TG:ABBRV+3. If the current tag word has more than four characters, it is not an article and control goes to TG:MC to check for 'M', 'Mc', or 'Mac'. At TG:ART the English and foreign article tables are checked. If the word is an article in one of the tables, it is ignored. The pointers to the tag build area are reset and control goes to TG:COND. If the word is not an article, a branch is taken to TG:MC.

At TG:ABBRV, the previous character is checked. If it was not a '.', control goes to TG:MC. Otherwise at TG:ABBRV+3, the current tag word is checked to see if it is longer than eight characters. If so, the abbreviation is not replaced and a branch is taken to TG:CLR. If the word has less than eight characters, the abbreviation table, ABLIST, is searched. If the word is not found in the

abbreviated list, a branch is taken to TG:CLR. If the word is found, it is replaced in the tag build area by its unabbreviated form. Then control goes to TG:CLR.

At TG:MC the current tag word is checked to see if it begins with 'Mc'. If not, a branch is taken to TG:CLR. If so, an 'a' is inserted between the 'M' and the 'c' and the rest of the word is adjusted in the tag build area. At TG:CLR the current tag word area (WD) is cleared to blanks and a branch is taken to TG:NUM to store the current character in the tag build area.

When the source string is exhausted, the rest of the tag build area is filled with blanks and control goes to TG:FUL. If at any point during the building process the tag area is filled, control goes to TG:FUL where the length of the compressed tag is calculated and stored in TG:LEN. Then at TG:COMPRO the tag bytes are compressed into 6-bit representation and replaced in the tag build area, four compressed bytes for every five regular tag bytes. When all the tag bytes are compressed, control is returned to the calling routine.

II. SOFTWARE INTERFACE

A. Linkage

```
BAL,R7   FMT:TAG
```

B. Parameter List Description

Upon entry to FMT:TAG R4 and R6 should be set as follows:

R4 - points to the unpacked text
R6 - points to the end of the unpacked text

C. Return Codes - none

D. Other Entry Points - none

E. OCLC CCFP Routines Referenced - none

F. OCLC Procedures Referenced

SAVE
RESTORE

I. FUNCTIONS

FMT:CDNO unpacks the Library of Congress card number into the bottom line of the catalog card (authority line). The byte address of the LC ared number in the input is picked up from P:CARDNO. The trailing digits (number portion) are located and leading zeros are scanned off. If no significant digits are found in the trailing portion, control is returned to the calling routine. When the first significant digit is encountered, the length of the trailing portion of the LC card number is saved. Then the trailing digits are unpacked into character words using the table UNPK:TBL, and the character words are stored left to right beginning in the far right position of the line.

When all trailing digits have been unpacked, a hyphen is stored to the left of the last unpacked digit. Then the two digit year and three preceding alphas are unpacked, continuing left to right. Imbedded blanks are deleted. Next the suffix length is checked. If the suffix will fit in the remaining card number space (maximum 13 positions), the portion of the card number already formatted is moved to the farthest left position that the LC card number may occupy in the authority line. Then the suffix is unpacked left to right following the number portion of the LC card number.

If the suffix is too long to fit in the remaining space, it is truncated. If the remaining space is less than three positions, only the first character of the suffix is formatted. If the remaining space is greater than or equal to three, as much of the suffix is formatted as will fit in the remaining space. If none of the suffix will fit, control is returned. After the card number and as much of the suffix as will fit are unpacked, control is returned through R7.

II. SOFTWARE INTERFACE

A. Linkage

```
BAL,R7   FMT:CDNO
```

B. Parameter List Description

Before entering FMT:CDNO, the byte address of the LC card number must be stored in P:CARDNO.

C. Return Codes - none

D. Other Entry Points - none

E. OCLC CCFP Routines Referenced - none

F. OCLC Procedures Referenced

```
PUSH
PULL
```

I. FUNCTIONS

This routine will insert text fields into the bottom line (authority line) of the catalog card. Upon entry to FMT:BOTLN if R5 contains zeros, the text to be unpacked is from the input record. If R5 is nonzero, it contains the length of the source field. In this case the text is from the PDT and an end of field delimiter is inserted at the end of the field to be unpacked. Then the routine FMT:UNPK is entered to convert the packed text to character words. If there is a syntax error in the packed text, an error message is printed, and processing on the current record is terminated. Upon return from FMT:UNPK, if a field delimiter was inserted, the last byte of the packed text is restored and control is returned via R7.

II. SOFTWARE INTERFACE

A. Linkage

```
BAL,R7   FMT:BOTLN
```

B. Parameter List Description

Upon entry to FMT:BOTLN, general registers R1, R2, R3 and R5 are set as follows:

R1 - contains the byte address of packed text.
R2 - points to the first word position of the packed text.
R3 - equals the length of the unpacked text.
R5 - equals zero or the length of the source field if a field delimiter is needed.

C. Return Codes - none

D. Other Entry Points - none

E. OCLC CCPF Routines Referenced

FMT:UNPK

F. OCLC Procedures Referenced

SAVE
RESTORE

I. FUNCTIONS

FMT:ORDER builds a complete sort tag using the sort tag ordering code in the PDT. Depending on the institution, the sort tag may be composed of the pack I.D., the shelf list pack I.D., the symbol 'NUC' or the cataloging library code and a combination of the tags built by FMT:TAG for the main entry, added entries, title or uniform title, the call number tag, the accession number, the LC card number or a combination of the holding library code and the LC card number.

Upon entry to FMT:ORDER, the tag area in the output buffer is cleared to zero. Then the sort order code is picked up from the PDT. If an error has occurred during processing of this card (MINOR:SW is greater than or equal to zero), the last option in both ORD:TBL1 and ORD:TBL2 will be used in building the sort tag. Otherwise the PDT sort order code is used to pick up the address of the routine to order the first part of the sort tag. The first part of the tag may consist of the pack I.D., the shelf list pack I.D., the symbol 'NUC' or the cataloging library code. In each of the above cases except for the NUC symbol, the desired I.D. is picked up from the PDT and moved to the sort tag area. For the NUC symbol, the characters 'NUC' are moved to the sort tag area. Then the sort tag area pointers are incremented and the address of the routine to order the remaining part of the sort tag is picked up from ORD:TBL2 using the sort tag ordering code from the PDT as a displacement.

If the second part of the sort tag is to be made up of the added entry tags, the routine ORD2:0 is entered. If the heading structure code, HD:SPEC, is one, the card is a shelf list or main entry card and control goes to ORD2:1 to use the main entry and uniform title or title tags. Otherwise 43 bytes of the heading tag are moved to the sort tag area followed by 24 bytes of the main entry tag. The sort tag pointers are incremented and control goes to ORD:XIT.

If the second part of the sort tag is to be made up of the main entry and uniform title (or title) tags, the routine ORD2:1 is entered. The first 43 bytes of the main entry tag are moved to the sort tag area followed by the first 24 bytes of the title tag. Then the sort tag area pointers are incremented and control goes to ORD:XIT.

If the second part of the sort tag is to be the call number tag, the routine ORD2:2 is entered. The four words which compose the alpha portion of the call number tag are stored in the sort tag area followed by the one word numeric portion of the tag and the fraction portion. The sort tag area pointers are incremented and a branch is taken to ORD:XIT.

If the accession number is to be the second part of the sort tag, the routine ORD2:3 is entered. If no accession number is present in the input record, a branch is taken to ORD:XIT and no accession number is put in the sort tag. Otherwise the packed accession number is moved into the sort tag area followed by the title tag. Then the sort tag area pointers are incremented and control goes to ORD:XIT.

If the LC card number is to be the second part of the sort tag, the routine ORD2:4 is entered. Then the packed LC card number from the input record is moved into the sort tag area. The sort tag area pointers are incremented and control goes to ORD:XIT.

If the holding library code and the LC card number are to be the second part of the sort tag, the routine ORD2:6 is entered. The holding library code is moved to the sort tag area from the PDT. Then control falls through to ORD2:4 where the LC card number will be moved in.

At ORD:XIT the registers are restored and the output buffer pointer and displacement are adjusted by the size of the sort tag (18 words). Then control is returned to the calling routine.

II. SOFTWARE INTERFACE

A. Linkage

```
BAL,R7    FMT:ORDER
```

B. Parameter List Description

Upon entry to FMT:ORDER R2, R3, and R4 should be initialized as follows:

R2 = WA(tag insertion area in output buffer)
R3 = maximum size of tag insertion area
R4 = 0

C. Return Codes - none

D. Other Entry Points - none

E. OCLC CCFP Routines Referenced - none

F. OCLC Procedures Referenced

SAVE
RESTORE

I. FUNCTIONS

The purpose of this routine is to move a seppacked paragraph (SPP) into the output record buffer. Upon entry to FMT:INSP, the call number length is Word 2 of the SPP leader is checked. If it is zero, no call number is associated with the paragraph and a branch is taken to INSP:10 to insert the SPP. Otherwise the call number is moved to the next position in the output buffer and the output buffer pointers are incremented. Then at INSP:10 the SPP is moved to the next position in the output buffer. The output buffer pointers are incremented, and control is returned to the calling routine.

If the end of theoutput buffer is reached, an error is declared and processing for this card request is terminated.

II. SOFTWARE INTERFACE

A. Linkage

```
BAL,R7    FMT:INSP
```

B. Parameter List Description

Upon entry to FMT:INSP R1, R2, R3 should be set up as follows:

R1 - WA(SPP leader)
R2 - P'(target area in output buffer)
R3 - LEN (target area)

C. Return Codes - none

D. Other Entry Points - none

E. OCLC CCFP Routines Referenced - none

F. OCLC Procedures Referenced

SAVE
RESTORE

I. FUNCTIONS

FMT:PRT is a generalized print routine with alternate entry points to print headings, move in text and clear the print buffer. At its initial entry point FMT:PRT, the lines per page count is checked and if necessary, headings are printed on a new page. Then the line of text passed by the calling program is moved into the print buffer and printed. The return is via a B 1,R7.

At the alternate entry point PRT:CLR the print buffer is cleared to blanks. The carriage control is reset for double spacing and control is returned through R6.

At PRT:HEAD, another alternate entry point, the page heading, page number, and date are printed. The line count is reset and the return is via a B 0,R6.

At PRT:MOVE the text string indicated by R4 is moved one byte at a time to the position in the print buffer indicated by R5. The return is through R6.

II. SOFTWARE INTERFACE

A. Linkage

The calling sequence for FMT:PRT is the following:

```
BAL,R7      FMT:PRT
DATA        BA(TEXTC)
```

B. Parameter List Description

The address of the line of text to be printed must be set up immediately following the BAL instruction to FMT:PRT as shown above. The length of the text string is assumed to be in the first byte of the string.

C. Return Codes - none

D. Other Entry Points

PRT:CLR
PRT:MOVE
PRT:HEAD

E. OCLC Subroutines Referenced - none

F. OCLC Procedures Referenced

PUSH
PULL

II. SOFTWARE INTERFACE (PRT:CLR)

A. Linkage

```
BAL,R6    PRT:CLR
```

B. Parameter List Description - none

C. Return Codes - none

D. Other Entry Points - none

E. OCLC Subroutines Referenced - none

F. OCLC Procedures Referenced - none

II. SOFTWARE INTERFACE (PRT:HEAD)

A. Linkage

BAL,R6 PRT:HEAD

B. Parameter List Description - none

C. Return Codes - none

D. Other Entry Points - none

E. OCLC Subroutines Referenced - none

F. OCLC Procedures Referenced

PUSH
PULL

II. SOFTWARE INTERFACE (PRT:MOVE)

A. Linkage
BAL,R6 PRT:MOVE

B. Parameter List Description

Before branching to PRT:MOVE, R3, R4, R5 should be set up as follows:

R3 - equals length of text to be moved.

R4 - points to the source string.

R5 - points to the target string.

C. Return Codes - none

D. Other Entry Points - none

E. OCLC CCFP Routines Referenced - none

F. OCLC Procedures Referenced - none

I. FUNCTIONS

FMT:ERR is a generalized routine to print error and warning diagnostics. If a new page is needed, the routine PRT:HEAD is used to print headings and page number. The routine PRT:CLR is used to clear the print buffer. If the error message is greater than fifty-three (53) characters, the message is truncated to 53 characters. If the message is an error rather than a warning, the number of rejected records is incremented; and the records associated with the current produce request are backspaced over and wiped off the output tape.

The input record number of the bad record is converted to EBCDIC to be printed. If no PDT is present for the record, the message is printed and control is returned via a B 1,R7. If a PDT is present, the OCLC number, the PDT number, institution, cataloging library, holding library, and receiving catalog are printed along with the massage. Then control is returned.

II. SOFTWARE

A. Linkage

The calling sequence is as follows:

```
BAL,R7      FMT:ERR
DATA        BA(ERRMSG) (Text with count)
```

B. Parameter List Description

The byte address of the message to be printed must be defined immediately following the BAL instruction as shown above.

C. Return Codes - None

D. Other Entry Points - none

E. OCLC CCFP Routines Referenced

```
PRT:HEAD      (FMT:PRT)
PRT:CLR       (FMT:PRT)
PRT:MOVE      (FMT:PRT)
BIN:BCD
```

F. OCLC Procedures Referenced

```
PUSH
PULL
@BINBCD
```

I. FUNCTIONS

FMT:RDTY issues a CAL1,1 to read from the console typewriter. It then loads R6 with the first word of the input message and returns control via a B 0,R7.

II. SOFTWARE INTERFACE

A. Linkage

```
BAL,R7   FMT:RDTY
```

B. Parameter List Description - none

C. Return Codes - Upon return R6 contains the first word of the input message.

D. Other Entry Points - none

E. OCLC CCFP Routines Referenced - none

F. OCLC Procedures Referenced - none

I. FUNCTIONS

FMT:WTTY will write a message to the console typewriter. It sets up the message address and message length in the FPT for the write CAL. The message length in BYTE0 of the message is blanked out for printing and restored before returning to the calling routine. Return is via a B 1,R7.

II. SOFTWARE INTERFACE

A. Linkage

The calling sequence is the following:

```
BAL,R7      FMT:WTTY
DATA        WA(Message) (Text with count)
```

B. Parameter List Description

The word address of the message to be printed must be defined in a word immediately following the BAL instruction as shown above.

C. Return Codes - none

D. Other Entry Points - none

E. OCLC CCFP Routines Referenced - none

F. OCLC Procedures Referenced -

PUSH
PULL

I. FUNCTIONS

This routine issues a CAL1,1 to read a card input transaction record. This routine is used primarily for testing data. When a read is issued, the count of input records is incremented. Control is returned to the calling routine via R7.

AT its alternate entry point, FMT:CHKRD, the read completion is checked. If the read is not complete, a wait is issued (CAL2,9 0) for the completion of the read. Upon completion, if an error occurred, a message is printed and processing is terminated via a CAL1,9 3 (abort). If an end of field or an end of data was encountered, control is returned to the calling routine via a B 0,R7. If the read was normal data, control is returned via a B 1,R7, where the switch, GOTRECSW, is incremented by two.

II. SOFTWARE INTERFACE

A. Linkage

The calling sequence is:

```
BAL,R7   FMT:RDX
```

B. Parameter List Description

Upon entry, the table P:XREC must be set up. The first word must be the byte address of the input and the third word must be the byte length of the input.

C. Return Codes -

The third word of the FPT contains the return code:

=0 Read is not complete
=1 Normal completion
=7 End of file encountered
=6 End of data encountered

D. Other Entry Points

FMT:CHKRD

E. OCLC CCFP Routines Referenced - none

F. OCLC Procedures Referenced - none

II. SOFTWARE INTERFACE

A. Linkage

```
BAL,R7    FMT:CHKRD
B         EOF
MTW,2     GOTRECSW
```

B. Parameter List Description

Upon entry, RDX:S contains the return code from the read.

C. Return Code

Upon return, P:XREC is set up for the next read.

D. Other Entry Points - none

E. OCLC CCFP Routines Referenced

FMT:ERR

F. OCLC Procedures Referenced

PUSH
PULL
@ERROR

I. FUNCTIONS

This routine issues a CAL3,0 to read a Profile Definition table (PDT) from disk. Upon entry to FMT:RPDT the completion code from the last PDT read is interrogated. If it is not posted as complete, a wait is issued (CAL2,9 0) for the completion of the read. (If a formatting error occurred, no wait may have been previously issued.) Before the read is issued the PDT number from the input record, PDT:NO, is compared to the PDT number currently in the PDT buffer. If they are equal, the request PDT is already in core. The switch, GOTPDTSW, is incremented by two, and control is returned to the calling routine. Otherwise PDT:NO is set up in the read FPT as the starting block number and the CAL#,0 is issued to read the PDT. Then control is returned to the calling routine.

At its alternate entry point, FMT:CHKPDT, the read completion is checked. If the read is not complete, a wait is issued. Upon completion if an error occurred, a message is printed and processing is terminated for the current card request. Otherwise the PDT is moved from the read buffer to the user-supplied buffer. Then the switch, GOTPDTSW, is incremented by two and control is returned to the calling routine.

II. SOFTWARE INTERFACE

A. Linkage

The calling sequence is:

```
BAL,R7   FMT:RPDT
DATA     PDT:L
```

B. Parameter List Description

Upon entry R7 must point to the PDT buffer as shown above. Also PDT:NO must contain the PDT number desired.

C. Return Codes - none

D. Other Entry Points

FMT:CHKPDT

E. OCLC CCFP Routines Referenced - none

F. OCLC Procedures Referenced

PUSH
PULL

III. SOFTWARE INTERFACE (EP: FMT:CHKPDT)

A. Linkage

```
BAL,R7   FMT:CHKPDT
```

B. Parameter List Description - none

C. Return Codes - none

D. Other Entry Points - none

E. OCLC CCFP Routines Referenced - none

F. OCLC Procedures Referenced

PUSH
PULL

I. FUNCTIONS

This routine issues a CAL1,1 to write the formatted card on the BO device. When a write is issued, the total number of bytes in the buffer is incremented by the length of this record. Control is returned to the calling routine via R7 where the switch, CRDRECNO, is incremented to indicate the number of records in this card set.

At its alternate entry point, FMT:WXWEOF, an end of file mark is written on the output device. If the write issued at FMT:WRTX is not complete, a wait is issued (CAL2,9 0) for the completion of the write. Upon completion, if an error occurred, a message is written on the operator's console and processing is terminated via a CAL1,9 3 (abort). Control is returned to the calling routine via R7.

At its alternate entry point, FMT:WXREW, a CAL1,1 is issued to rewind the BO device.

At its alternate entry point, FMT:WXUNL, a CAL1,1 is issued to unload the BO device.

II. SOFTWARE INTERFACE

A. Linkage

The calling sequence is:

```
BAL,R7   FMT:WRTX
```

B. Parameter List Description

Upon entry, the table OP:REC must be set up. The first word must be the byte address of the formatted card field. The third word must be the length of the formatted card field. The second and fourth words are the maximum word length of the output buffer.

C. Return Codes - the third word of the FPT will contain a zero if the write was not complete. It will contain a one for a normal completion. Anything else is considered an error.

D. Other Entry Points

```
FMT:WXWEOF
FMT:WXREW
FMT:WXUNL
```

E. OCLC CCFP Routines Referneced - none

F. OCLC Procedures Referenced

```
PUSH
PULL
@ERROR
@WTTY
```

II. SOFTWARE INTERFACE

A. Linkage

The calling sequence is:

```
BAL,R7  FMT:WXWEOF
```

B. Parameter List Description

Upon entry, WRTX:S contains the return code from the write issued by FMT:WRTX.

C. Return Codes - none

D. Other Entry Points - none

E. OCLC CCFP Routines Referenced - none

F. OCLC Procedures Referenced

```
PUSH
PULL
@ERROR
@WTTY
```

II. SOFTWARE INTERFACE

A. Linkage

The calling sequences are:

```
BAL,R7    FMT:WXREW
BAL,R7    FMT:UNL
```

B. Parameter List Description - none

C. Return Codes - none

D. Other Entry Points - none

E. OCLC CCFP Routines Referenced - none

F. OCLC Procedures Referenced - none

I. FUNCTIONS

The purpose of BIN:BCD is to convert a twenty-four bit binary value to EBCDIC inserting a decimal point if required. BIN:BCD converts the input binary value and stores it in the target area. After the maximum number of decimal digits has been converted, the first byte of the binary value word is checked to see if any decimal digits were desired. If so, the right most decimal digits are moved to the right and a decimal point is inserted. Control is returned to the calling routine via R7.

II. SOFTWARE INTERFACE

A. Linkage

The calling sequence for BIN:BCD is

```
BAL,R7   BIN:BCD
DATA     WA(BIN)
DATA     BA(BCD)
```

B. Parameter List Description

Parameters are passed to BIN:BCD as described above.

BIN is a word of which eight bits is the number of significant digits which will go to the right of a decimal point. The remaining twenty-four bits contain the binary value to be converted.

The maximum number of decimal digits is six (6).

C. Return Codes - none

D. Other Entry Points - none

E. OCLC CCFP Routines Referenced - none

F. OCLC Procedures Refenced

PUSH
PULL

I. FUNCTIONS

The purpose of BIN:BCDR is to convert a variable length binary field to EBCDIC, right justify it, and move it to the target area. Control is returned to the calling routine via R7.

II. SOFTWARE INTERFACE

A. Linkage

The calling sequence for BIN:BCDR is

```
LI,R3     BA(Target End)
LW,R5     Binary Value
LI,R6     Target Length
BAL,R7    BIN:BCDR
```

B. Parameter List Description

Parameters are passed to BIN:BCDR as described above.

C. Return Codes

If the first character of the binary field indicated by R5 is zero, the target area is filled with a string of asterisks (*).

D. Other Entry Points - none

E. OCLC CCFP Routines Referenced - none

F. OCLC Procedures Referenced - none

I. FUNCTIONS

BCD:HEX is a generalized routine to convert a 4 byte field of alphas in EBCDIC format to Binary format. The field to be converted is passed in R6. The converted field is returned in R6. If the field was not alpha, R6 will be 0.

II. SOFTWARE INTERFACE

A. Linkage

The calling sequence is as follows:

```
BAL,R7   BCD:HEX
```

B. Parameter List Description

The word to be converted must be in R6.

C. Return Codes

R6 will contain the converted field. If the field was not alpha, R6 will be 0.

D. Other Entry Points - none

E. OCLC CCFP Routines Referenced - none

F. OCLC Procedures Referenced

```
PUSH
PULL
```

I. FUNCTIONS

BCD:BIN is a generalized routine to convert a 4 byte field of numerics in EBCDIC format to Binary format. The field to be converted is passed to R6. The converted field is returned in R6. Return is via a B 0,R7. If the field is not numeric, R6 will be 0.

II. SOFTWARE INTERFACE

A. Linkage

The calling sequence is as follows:

```
BAL,R7 - BCD:BIN
```

B. Parameter List Description

The word to be converted must be in R6.

C. Return Codes- R6 will contain the converted field. If the field was not numeric, R6 will be 0.

D. Other Entry Points - none

E. OCLC CCFP Routines Referenced - none

F. OCLC Procedures Referenced

```
PUSH
PULL
```

APPENDIX E

ADDITIONAL PROCEDURE DOCUMENTATION

PROCEDURE DESCRIPTION

PURPOSE:

The purpose of the PUSH/PULL procedure is to save and rstore registers bv pushing them into and pulling them from the push down stack defined bv :PCBPT.

There are two alternate naems for this procedure, PUSH and PULL. The CNAME value assigned to each name indicates whether to generate the code for a PUSH or a PULL instruction.

PUSH has a CNAME value of 1. If there is a single value in the argument field, a PSW instruction is generated. If there are two values in the argument field, a PSM instruction is generated.

PULL has a CNAME value of 0. If there is a single value in the argument field, a PLW instruction is generated. If there are two values in the argument field, a PLM instruction is generated.

FORMAT:

```
PUSH    RX
PUSH    RX,RY
PULL    RX
PULL    RX,RY
```

EXAMPLE 1:

```
PUSH    R1
```

The procedure will push R1 into the push down stack.

EXAMPLE 2:

```
PULL    R1, R7
```

The procedure will generate an LCI 7 instruction and will pull seven registers from the stack starting with R1.

PROCEDURE DESCRIPTION

PURPOSE:

The purpose of the SAVE/RESTORE procedure is to save or restore the registers via a store or load instruction. The second or third argument field must be the address of a save area.

There are two alternate names for this procedure, SAVE and RESTORE. The CNAME value assigned to each name indicates whether to place the registers into or load the registers from the save area.

SAVE has a CNAME value of 1. If the third argument field is missing, it causes the assembly of an instruction in the form of:

```
STW,AF(1)  AF(2)
```

If the third argument field is present, it causes the assembly of two instructions in the form of:

```
LCI          AF(2) -AF(1) +1
STM,AF(1)    AF(3)
```

RESTORE has a CNAME value of 0. If the third argument field is missing, it causes the assembly of an instruction in the form of:

```
LW,AF(1)     AF(2)
```

If the third argument field is present, it causes the assembly of two instructions in the form of:

```
LCI          AF(2) - AF(1) +1
LM,AF(1)     AF(3)
```

FORMAT:

```
SAVE      RX,SVA
SAVE      RX,RY,SVA
RESTORE   RX,SVA
RESTORE   RX,RY,SVA
```

EXAMPLE 1:

```
SAVE      R1,CC:SVA
```

The procedure will store R1 at location CC:SVA.

EXAMPLE 2:

```
RESTORE   R1,R7,CC:SVA
```

The procedure will load registers 1 - 7 from location CC:SVA.

PROCEDURE DESCRIPTION

PURPOSE:

@RDTY provides the linkage to the CCFP subroutine FMT:RDTY. If the argument field is greater than zero, the return code from FMT:RDTY will be stored at the location indicated by the argument field.

FORMAT:

```
@RDTY    AF(1)
```

EXAMPLE:

```
@WTTY    MSGSVA
```

The procedure causes the assembly of two instructions in the form of:

```
BAL,R7    FMT:RDTY
STW,R6    MSGSVA
```

Where R6 contains the first word of the input message.

PROCEDURE DESCRIPTION

PURPOSE:

@WTTY provides the linkage to the CCTF subroutine, FMT:WTTY. The argument field must be the word address of the message to be written, in the format of a text with count instruction.

FORMAT:

```
@WTTY    AF(1)
```

EXAMPLE:

```
@WTTY    MSG12
```

The procedure causes the assembly of two instructions in the form of:

```
BAL,R7    FMT:WTTY
DATA      MSG12
```

PROCEDURE DESCRIPTION

PURPOSE:

@PRINT provides the linkage to the CCFP subroutine, FMT:PRT. The argument filed must be the byte address of the line of text to be printed, in the format of a text with count instruction.

FORMAT:

```
@PRINT    AF(1)
```

EXAMPLE:

```
@PRINT    MSG12
```

The procedure causes the assembly of two instructions in the form of:

```
BAL,R7    FMT:PRT
DATA      BA(MSG12)
```

PROCEDURE DESCRIPTION

PURPOSE:

@BINBCD provides the linkage to the CCFP subroutine, BIN:BCD. The first argument field must be the word address of the binary field to be converted. The second argument field must be the byte address of the target field. The third argument field, if it is present, is the byte displacement into the target field, so that the sum of argument field two and three will give the byte address of where to store the converted field. The fourth argument field, if it is present, is the number of decimal digits required in the converted field.

FORMAT:

```
@BINBCD    AF(1), AF(2), AF(3), AF(4)
```

EXAMPLE:

```
@BINBCD    PTEMP,MSG1,20,2
```

The procedure causes the assembly of three instructions in the form of:

```
BAL,R7     BIN:BCD
GEN,8,24   2,PTEMP
DATA       BA(MSG1)+20
```

PROCEDURE DESCRIPTION

PURPOSE:

@BCDHEX provides the linkage to the CCFP subroutine, BCD:HEX. If there is an argument field present, the converted Binary field is stored at this location.

FORMAT:

```
@BCDHEX   AF(1)
```

EXAMPLE:

```
@BCDHEX   HEXSVA
```

The procedure causes the assembly of two instructions in the form of:

```
BAL,R7    BCD:HEX
STW,R6    HEXSVA
```

PROCEDURE DESCRIPTION

PURPOSE:

@BCDBIN provides the linkage to the CCFP subroutine, BCD:BIN. If there is an argument field present, the converted Binary field is stored at this location.

FORMAT:

```
@BCDBIN / AF(1)
```

EXAMPLE:

```
@BCDBIN   BINSVA
```

The procedure causes the assembly of two instructions in the form of:

```
BAL,R7    BCD:BIN
STW,R6    BINSVA
```

PROCEDURE DESCRIPTION

PURPOSE:

@ERROR provides the linkage to the CCFP subroutine, FMT:ERR. Control is then passed to the subroutine FMT:INIT or to any other routine as indicated by the presence of a second argument field. The first argument field must be the byte address of the error message in the form of a text with count instruction.

FORMAT:

```
@ERROR    AF(1), AF(2)
```

EXAMPLE 1:

```
@ERROR    ERROR:1
```

The procecure causes the assembly of three instructions in the form of:

```
BAL,R7    FMT:ERR
DATA      BA(ERROR:1)
B         FMT:INIT
```

EXAMPLE 2:

```
@ERROR    ERROR:2, $+2
```

The procedure causes the assembly of three instructions in the form of:

```
BAL,R7    FMT:ERR
DATA      BA(ERROR:2)
B         $+2
```

PURPOSE:

The procedure 'ROMAN' creates the entries for the table V:ROMTBL. Each entry may have a maximum of 4 bytes where each byte represents one character of the Roman numeral for the numbers 1-10. If there is no argument field, it is flagged as an error.

FORMAT:

```
ROMAN   AF(1), AF(2), AF(3), AF(4)
```

EXAMPLE:

```
ROMAN   'V', 'I', 'I', 'I'
```

creates a one word entry in the table equivalent to the Roman numeral VIII.

PURPOSE:

The procedure 'RETURN' is used to generate a Branch instruction. Each RETURN command creates one word in the Macrsym instruction format. The first eight bits contain the hexadecimal code for a branch indirect on a register, X'E8'. The next seven bits are zero. The last 17 bits, [AF(1)], contain the reference address.

FORMAT:

```
RETURN    AF(1)
```

EXAMPLE:

```
RETURN    R14
```

Generates an instruction in the form 'E8000000E' which is effectively the same as B *R14.

PURPOSE:

The 'TBL' commands build the entries for the tag table (ITBL), the index table (ITBL), the processing pointer table (PTBL). These tables are used by the subroutine, FMT:MAP.

FORMAT:

_TBL AF(1), AF(2), AF(3), AF(4)

EXAMPLES:

Example 1:

TTBL 049

creates a half-word entry in the MAP:TBL table for the 049 field (user holdings information). If there is no argument field, it is flagged as an error.

Example 2:

ITBL 2

creates a 1 byte entry in the MAP:ITBL table which will be used as an index for the 049 field. If there is no agrument field, it is flagged as an error.

Example 3:

PTBL 0,1,P:CN

creates a 1 word entry in the MAP:PTBL table. The first 8 bits [AF(1)] are a unique statement identifier. The next 8 bits [AF(2)] indicate the relative column in the SPCR which specifies how the card associated with a particular type of entry is to be built. The next half-word [AF(3)] is the address of the pointer table where the address of the field is to be saved. That is, the address of the call number field will be saved at location P:CN. All three argument fields must be present.

PURPOSE:

The procedure 'OTBL' creates the entries for the table CC:OPTBL which is used by the subroutine, FMT:CC. This will create one 4-byte entry for each of the input fields (main entry, uniform title, etc.). Each byte is an index into each of the tables CC:PBF1, CC:PBF2, CC:PSF, and CC:PHF.

FORMAT:

```
OTBL    AF(1), AF(2), AF(3), AF(4)
```

PURPOSE:

The procedure 'OPTB' creates the entries for the OPTB:TBL table which provides the indicators for processing the heading. Each half-word entry in the table contains four indicators. The indicators are described on page VIII.4 with the description of OPTB:TBL. The first indicator is 3 bits, the second is 1 bit, the third is 4 bits, the fourth is 8 bits.

FORMAT:

```
OPTB    AF(1), AF(2), AF(3), AF(4)
```

EXAMPLE:

```
OPTB    3,1,0,17
```

creates a half-word entry for the dropped, stand-alone heading that is to be all upper-case with a normal structure.

PROCEDURE DESCRIPTION

PURPOSE:

The 'GEN' commands build the entries for the unpack table (UNPK:TBL). The 12 different commands and their corresponding purpose are:

GENA0	generate lower case alpha
GENA1	generate upper case alpha
GENN	generate a numeric
GENBLK	generate a blank
GEND	generate a diacritical
GENUND	generate an underscore
GENVH	generate vanishing hyphen (not used at present)
GENNUL	generate a null character
GENSPEC	generate special unpack (control command)
GEN:	generate a delimiter
GENNONE	generate a table full indicator (not used at present)
GENP	generate punctuation

FORMAT:

```
GEN    AF(1), AF(2)
```

EXAMPLES:

Example 1:

```
GENA0  X'81'
```

creates an entry in the table in form of X'80000081' which specifies the character word for a lower case 'a'.

Example 2:

```
GENA1  X'C1'
```

creates an entry in the table in the form of X'800000C1' which specifies the character word for an upper case 'A'.

Example 3:

```
GENN   X'F1'
```

creates an entry in the table in the form of X'400000F1' which specifies the character word for a numeric '1'.

Example 4:

```
GENBLK  X'40'
```

creates an entry in the table in the form of 'X30000040' which specifies the character word for a blank.

Example 5:

```
GEND    X'30'
```

creates an entry in the table in the form of X'00300000' which specifies the character word for the ligature diacritical.

Example 6:

```
GENVND  X'6D'
```

creates an entry in the table in the form of X'00008000' which specifies the character word for an underscore.

Example 7:

```
GENNUL  X'59'
```

creates an entry in the table in the form of X'00000000' which specifies a null or unprintable symbol.

Example 8:

```
GENSPEC 3
```

creates an entry in the table in the form of X'00028003' which specifies the control command for a minus or hyphen.

Example 9:

```
GEN:    ,3
```

creates an entry in the table in the form of X'00018003' which specifies the character word for an end of field delimiter.

Example 10:

```
GENP    X'AD',2
```

which creates an entry in the table in the form of X'200000AD' which specifies the punctuation character left bracket.

PURPOSE:

The 'GEN:' commands build the entries for the language type table; LANG:TBL. Each one word entry in the LANG:TBL has the word address of the corresponding language article table and the number of articles present in that particular table. Both AF(1) and AF(2) must be present for the GEN:LG2 command. Only AF(1) must be present for the GEN:LG1 command.

FORMAT:

GEN:LG2 AF(1), AF(2)

EXAMPLE:

GEN:LG2 LG:GER -1,5

creates a one word entry in the LANG:TBL table. The first 8 bits contain a '5' to indicate that there are 5 entries in German article table. The next 24 bits contain the word address of the German article table.

PURPOSE:

The procedure 'CALL' is used to generate a BAL instruction. Each CALL command creates one word in the Macrsvm instruction format. The first eight bits contain the hexadecimal code for a BAL instruction, X'6A'. The next four bits [AF(1)] determine which is the linking register. The next 3 bits are 0 and the last 17 bits [AF(2)] contain the reference address. Both AF(1) and AF(2) must be present.

FORMAT:

```
CALL   AF(1) , AF(2)
```

EXAMPLE:

```
CALL   R7,FMT:WXREW
```

Generages an instruction in the form of '6A701A74' which is effectively the same as BAL,R7 FMT:WXREW.

PURPOSE:

The 'B' procedure commands are used to set up special branch instructions. The procedures names and their corresponding functions are:

Procedure Name	Function
BALP	branch if alpha
BNALP	branch if not alpha
BNUM	branch if numeric
BNNUM	branch if not numeric
BPUN	branch if punctuation
BNPUN	branch if not punctuation
BLB	branch if potential left break
BNLP	branch if not potentional left break
BRB	branch if potential right break
BNRB	branch if not potential right break
BB	branch if blank
BNB	branch if not blank

Each procedure name creates one word in the Macrsym instruction format. The first bit, if turned on, specifies indirect addressing. The next seven bits contain the hexadecimal operation code for the type of branch required; X'69' for a BCS or X'68' for a BCR. The next four bits contain the condition code. The next 3 bits, [AF(2)], contain the index. The last seventeen bits, [AF(1)] which must be present, contain the reference address.

FORMAT:

```
BALP   AF(1), AF(2)
```

EXAMPLES:

Example 1:

```
BALP   $+3
```

This will assemble as '69801079' which is the instruction format for BCS,8 $+3 which effectively branches if alpha.

Example 2:

```
BNUM   VCN:NUM2
```

This will assemble as '69401081' which is the instruction format for BCS,4 VCN:NUM2 which effectively branches if numeric.

APPENDIX F

INTERNAL SUBROUTINE CROSS-REFERENCE

the ohio college library center
1314 kinnear rd. - columbus ohio - 43212

SUBROUTINE CROSS-REFERENCE

CALLING ROUTINE / ROUTINE REFERENCED	OCLCCCFP	FMT:MAP	FMT:BOTLN	FMT:ERR	SPK:CP	SPK:ERCN	FMT:UPCN	FMT:SPCN	FMT:UPCP	FMT:INSN	FMT:SPK	FMT:SPSP	FMT:CC	FMT:RDX	FMT:WRTX	
FMT:WTTY	X														X	
FMT:RDX	X															
FMT:WRTX	X															
FMT:RPDT	X															
FMT:MAP	X															
FMT:CDNO	X															
FMT:HSV	X															
FMT:UPCN	X															
FMT:CC	X								X							
FMT:LKCN	X															
AE:1	X															
SP:1	X															
FMT:TAG	X															
FMT:BOTLN	X															
FMT:CAP	X															
FMT:DCAP	X															

the ohio college library center
1314 kinnear rd. - columbus ohio - 43212

SUBROUTINE CROSS-REFERENCE

CALLING ROUTINE / ROUTINE REFERENCED	OCLCCCFP	FMT:MAP	FMT:BOTLN	FMT:ERR	SPK:CP	SPK:ERCN	FMT:UPCN	FMT:SPCN	FMT:UPCP	FMT:INSN	FMT:SPK	FMT:SPSP	FMT:CC	FMT:RDX	FMT:WRTX
FMT:SPK	X											X			
FMT:ERR	X													X	X
FMT:ORDER	X														
FMT:INSP	X									X					
FMT:SPSP	X														
FMT:SP1	X										X				
FMT:SP2	X										X				
FMT:INSN	X														
FMT:PRT	X														
BIN:BCD	X			X											
BIN:BCDR		X		X											
FMT:UPCP					X										
SPK:CP						X					X				
FMT:UNPK			X				X						X		
FMT:SPTS								X				X			
FMT:SPCN											X				
SPK:ERCN											X				

the ohio college library center
1314 kinnear rd. - columbus ohio - 43212

SUBROUTINE CROSS-REFERENCE

CALLING ROUTINE / ROUTINE REFERENCED	AE:1	SP:1															
FMT:WTTY																	
FMT:RDX																	
FMT:WRTX																	
FMT:RPDT																	
FMT:MAP																	
FMT:CDNO																	
FMT:HSV	X																
FMT:UPCN																	
FMT:CC	X	X															
FMT:LKCN																	
AE:1																	
SP:1																	
FMT:TAG																	
FMT:BOTLN																	
FMT:CAP																	
FMT:DCAP																	

the ohio college library center
1314 kinnear rd. - columbus ohio - 43212

SUBROUTINE CROSS-REFERENCE

CALLING ROUTINE / ROUTINE REFERENCED	ABEND	ABEND																
FMT:SPK																		
FMT:ERR																		
FMT:ORDER																		
FMT:INSP																		
FMT:SESP																		
FMT:SP1																		
FMT:SP2																		
FMT:INSN																		
FMT:PRT																		
BIN:BCD																		
BIN:BCDR																		
FMT:UPCP																		
SPK:CP																		
FMT:UNPK																		
FMT:SPTS																		
FMT:SPCN																		
SPK:ERCN																		